Geologically Correct Terrain Data Structures and Radar Siting
HM1582-05-2-0002

Dr W Randolph Franklin, Professor
ECSE Dept, 6026 JEC
Rensselaer Polytechnic Institute
110 8th St
Troy NY 12180

telephone: +1 518 276 6077
fax: (ask)
email: mail@wrfranklin.org or wrf@ecse.rpi.edu or franklin@cs.rpi.edu or frankwr@rpi.edu

March 9, 2009

Contents

1 Final Report 5
2 Press Releases 29
3 Papers and Public Presentations 38

2.1 Tradeoffs when multiple observer siting on large terrain cells – SDH 2006 paper 51
2.2 Tradeoffs when multiple observer siting on large terrain cells – SDH 2006 talk 66

3.1 Two novel surface representation techniques – Autocarto 2006 40
3.2 Compressing terrain datasets using segmentation – SPIE 2006 93
3.3 Terrain representation using tessellation of irregular planar tiles – FWCG 2006 poster 99
Terrain representation using tessellation of irregular planar tiles – FWCG 2006 abstract 100

Multiple observer siting on a compressed terrain – FWCG 2006 poster 102

Multiple observer siting on a compressed terrain – FWCG 2006 abstract 103

An improved LLL algorithm – Linear Alg & Apps 105

Surface compression using over-determined Laplacian approximation — SPIE 2007 117

Path planning on lossily compressed terrain — SPIE 2007 129

Smugglers and border guards – the GeoStar project at RPI – ACMGIS 2007 141

Smugglers and border guards – the GeoStar project at RPI – slides – ACMGIS 2007 149

Drainage network and watershed reconstruction on simplified terrain – FWCG 2007 abstract 167

Drainage network and watershed reconstruction on simplified terrain – FWCG 2007 poster 169

Approximating terrain with over-determined Laplacian PDEs – FWCG 2007 abstract 170

Approximating terrain with over-determined Laplacian PDEs – FWCG 2007 poster 172

Slope accuracy and path planning on compressed terrain — SDH 2008 173

Slope accuracy and path planning on compressed terrain — SDH 2008 talk 188

Progressive transmission of lossily compressed terrain — CLEI 2008 202

Efficient viewshed computation on terrain in external memory — 2008 212

Path planning on complex terrain – FWCG 2008 abstract 1 237

Path planning on complex terrain – FWCG 2008 talk 1 239

Path planning on complex terrain – FWCG 2008 poster 1 249

Operating on large geometric datasets – FWCG 2008 abstract 2 250

Operating on large geometric datasets – FWCG 2008 talk 2 252

Parallel ODETLAP for terrain compression and reconstruction — ACMGIS 2008 paper 1 265

Parallel ODETLAP for terrain compression and reconstruction — ACMGIS 2008 talk 1 274

Path planning on a compressed terrain — ACMGIS 2008 paper 2 290
Oct 2007 review: DEM compression and terrain approximation – smugglers and border guards

Nov 2007 Slope compression retasking

Feb 2008 Task Summary

Geo* at RPI – Jun 2008 site visit

Hydrology aware triangulation of terrain data – Jun 2008 site visit poster

Hydrology aware triangulation of terrain data – Jun 2008 site visit talk

Path Planning and Slope Representation of a Compressed Terrain – Jun 2008 site visit

Aug 2008 review: Geo* at RPI

5 Doctoral Thesis

Compressing terrain elevation datasets – Metin Inanc

6 Masters Theses

Evaluating and compressing hydrology on simplified terrain – Jon Muckell

Representation, compression and progressive transmission of digital terrain data using over-determined Laplacian partial differential equations – Zhongyi Xie

Parallel terrain compression and reconstruction – Jake Stookey
1 Final Report
1. Introduction

2. Personnel
   2.1. Faculty
   2.2. Contractor
   2.3. Students

3. Task summary
   3.1. In proposal
   3.2. Tasks chosen by NGA -- Accomplished
   3.3. Post-award modifications
   3.4. Tasks accomplished
      3.4.1. Morphological terrain sculpting
      3.4.2. Overdetermined Laplacian PDE (ODETLAP) representation
      3.4.3. Triangulated Irregular Network representation
      3.4.4. Siting/intervisibility toolkit
      3.4.5. Missing data fillin
      3.4.6. Lossy compression by a factor of 100
      3.4.7. Path planning or Motion planning
      3.4.8. Slope accuracy in lossy compression
      3.4.9. Parallel ODETLAP
3.4.10. Evaluating hydrology preservation during compression

4. Value of this project to DARPA

5. Videos

6. Software

7. Press and Blog Mentions
   7.1. Inside RPI
   7.2. Outside RPI

8. Publications / Public Presentations

9. Commercialization

10. Followon Projects

1. Introduction

We have the pleasure to present the final report for the DARPA/Geo* project at RPI, Geologically Correct Terrain Data Structures and Radar Siting. This report consists of a large PDF file, several tarballs of software and several videos. They are all available on the password-protected web site

http://www.ecse.rpi.edu/~wrf/wiki/GeoStar/2009-02-final/

Since the PDF file is about 900 pages and 150MB, it is available both as one file

RPI-GeoStar-final-report.pdf

and split up into nine 100-page parts.


2. Personnel

2.1. Faculty
<table>
<thead>
<tr>
<th>Who</th>
<th>Title</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr Marcus A V ANDRADE</td>
<td>Assistant Professor, Universidade Federal de Viçosa, Brazil</td>
<td>Visited RPI from April 2007 to April 2008 sponsored by the government of Brazil. Continuing to collaborate after his return.</td>
</tr>
<tr>
<td>Dr Barbara M CUTLER</td>
<td>Assistant Professor, Computer Science Department, RPI</td>
<td>Computer Graphics expert, assistant director of GeoStar project at RPI</td>
</tr>
<tr>
<td>Dr W Randolph FRANKLIN</td>
<td>Professor, ECSE Dept, RPI</td>
<td>PI</td>
</tr>
<tr>
<td>Dr Frank LUK</td>
<td>Professor, Computer Science Dept, RPI</td>
<td>Numerical analysis expert; moved to Hong Kong to be Vice-President (Academic) of Hong Kong Baptist University</td>
</tr>
<tr>
<td>Dr Clark K RAY III</td>
<td>USMA</td>
<td>terrain visibility</td>
</tr>
<tr>
<td>Dr Caroline WESTORT</td>
<td>Research Assistant Professor, ECSE, RPI</td>
<td>scooping; contract ended Dec 2006.</td>
</tr>
</tbody>
</table>

**2.2. Contractor**

<table>
<thead>
<tr>
<th>Who</th>
<th>Title</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe ROUBAL</td>
<td>ESRI ArcGIS siting DLL; subcontract is finished.</td>
<td></td>
</tr>
</tbody>
</table>

**2.3. Students**

<table>
<thead>
<tr>
<th>Who</th>
<th>Title</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr Metin INANC</td>
<td>Research Assistant, CS Dept, RPI.</td>
<td>ODETLAP etc. Graduated, May 2008 with Ph.D.</td>
</tr>
<tr>
<td>You LI</td>
<td>Teaching Assistant and PhD student, CS Dept, RPI</td>
<td>started Fall 2008, bathymetry</td>
</tr>
</tbody>
</table>
### Jon MUCKELL
Teaching Assistant, ECSE Dept, RPI. Graduated May 2008 with M.Sc.

### Jake STOOKEY
Masters student
Part-time work on adapting ODETLAP to the massively parallel IBM BG/L machine in RPI's CCNI.

### Chris STUETZLE
Research Assistant and PhD student, CS Dept, RPI
Fitting slopes with ODETLAP. Started May 2008.

### Dan TRACY
Research Assistant and PhD student, CS Dept, RPI
Path planning and siting

### Zhongyi XIE
Research Assistant, CS Dept, RPI.
ODETLAP etc. Graduated, May 2008 with M.Sc.

### Eddie Lau Tsz YAM
Teaching Assistant and PhD student, CS Dept, RPI
started Fall 2008, bathymetry

## 3. Task summary

### 3.1. In proposal

Our proposal listed the following tasks

1. Terrain representation
   a. Morphological terrain sculpting
   b. Overdetermined Laplacian PDE (ODETLAP)
   c. Triangulated irregular network (TIN)
   d. Lossy compression

2. Terrain operators
   a. Siting/intervisibility toolkit
   b. Trajectory planning
   c. Drainage analysis

### 3.2. Tasks chosen by NGA -- Accomplished
NGA listed the following specific tasks for us to accomplish:

1. In phase I, compress terrain by a factor of 10 with reasonable error.
2. In phase II, compress terrain by a factor of 100.

We accomplished the phase II task by Oct 2007, and presented our results at that review. The following graph shows the results on our six sample data sets. In each case, the uncompressed binary file size was 320KB. In each case, the uncompressed ASCII file size was from 800KB to 2400KB. We feel that computing compression ratios conservatively, relative to the binary file size is more useful. However, since some other researchers under the Geo* program use the ASCII file size, we include that also for your convenience.

The graph shows that we compressed the datasets by a factor of 100 while achieving an RMS elevation error ranging from 1% to 5%, depending on how mountainous the data set was.

### 3.3. Post-award modifications

Since the original award, there were several occasions at which we were instructed by Dr Carey Schwartz to modify our goals. This generally occurred
at the program review meetings, either during our presentation or in personal discussions after. Here are the most important changes.

1. **Make path planning an important theme.**

   We were given these instructions at the April 2006 review.

   In response, we started a major effort in this area, with significant results. We can compute paths on hi-res terrain around complex forbidden zones, such as the viewsheds of multiple well-placed observers.

   ![Path avoiding many observers](image)

1. **Don't productize so much** *and* **Do not award the remaining budgeted money to ESRI.**

   In response, we did not make the 2nd part of the award to ESRI listed in the original budget. The money was mostly used to hire Dr Caroline Westort as a research assistant professor for 20 months.

   We also stopped spending time cleaning and documenting our code, and used the time to produce more results.
2. Concentrate on our best terrain representation.

In response, we stopped working on morphological terrain operators like scooping, although that area still has the greatest long term potential. We ended Dr Westort's contract, since that was her speciality. Thereafter, We used only our ODETLAP representation, since it was the most mature (and yet has considerable potential).

1. Stop concentrating on extreme terrain compression. Rather,
concentrate on representing slope accurately.

We were given these instructions on 11/7/2007.

This major change was first mentioned at our only site visit to NGA, in St Louis. Later Dr Schwarz re-emphasized it. We were excited to work on this new topic, since there is no prior art. In addition, and contrary to intuitive, an accurate elevation representation does not at all guarantee slope accuracy. We have made major advances, described in detail in the attached papers and reports.

3.4. Tasks accomplished

Here is a summary of the tasks accomplished on this project. More details are in the attached presentations and papers.

3.4.1. Morphological terrain sculpting

We transferred the resources to more short-term projects.
3.4.2. Overdetermined Laplacian PDE (ODETLAP) representation

This was our major success, forming base of most of our current work. The key differentiating factors of ODETLAP are:

1. It produces a smooth surface, not showing the original data.
2. It allows for possible progressive transmission.
3. It can conflate inconsistent partially overlapping data sets.
4. It can interpolate partial sets of elevation posts.
5. It can infer a local maximum inside the topmost contour of a set of nested contours.

3.4.3. Triangulated Irregular Network representation

1. We can process $10^4 \times 10^4$ points on a laptop.
2. It finds the points in order of importance (unlike competing methods).
3. It works in core (unlike competitors).
3.4.4. Siting/intervisibility toolkit

ESRI produced an ArcGIS DLL. However, then development was ordered to be stopped.
3.4.5. Missing data fillin

One serendipitous application of our ODETLAP representation is the ability to fill in large regions of missing data. Responding to a request at an early review meeting, the following figure shows six interpolation methods for filling in a missing hole of radius 100. The top left image, which is ODETLAP, shows how local maxima in the missing region are inferred and realistic contours are generated. The top middle is a precursor to ODETLAP, and is slightly less realistic. The top right image is comparable to ODETLAP, but requires a higher order differential equation. The bottom three images show three Matlab techniques; all are quite unrealistic.
3.4.6. Lossy compression by a factor of 100

Done; details are in our reports and papers.

<table>
<thead>
<tr>
<th>Data</th>
<th>Size, bytes</th>
<th>Compression ratio</th>
<th>RMS Elev Error, m</th>
<th>RMS Slope Error, deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>1880</td>
<td>170:1</td>
<td>2.83</td>
<td>3.53</td>
</tr>
<tr>
<td>hill2</td>
<td>1962</td>
<td>163:1</td>
<td>4.06</td>
<td>8.06</td>
</tr>
<tr>
<td>hill3</td>
<td>1739</td>
<td>184:1</td>
<td>1.66</td>
<td>1.65</td>
</tr>
<tr>
<td>mtn1</td>
<td>1979</td>
<td>162:1</td>
<td>3.77</td>
<td>14.0</td>
</tr>
<tr>
<td>mtn2</td>
<td>2006</td>
<td>160:1</td>
<td>4.31</td>
<td>14.1</td>
</tr>
<tr>
<td>mtn3</td>
<td>2004</td>
<td>160:1</td>
<td>4.58</td>
<td>13.3</td>
</tr>
</tbody>
</table>
Accuracy of ODETLAP compression by more than 100:1

Note that our base for compression ratio comparison is the original binary file size. Some others use the original ASCII file size, which is much larger.

The slope accuracies are before we customized ODETLAP for slopes.

3.4.7. Path planning or Motion planning

1. We can do this on Level-II DEMs,

2. and on urban LIDAR data.

3. Our method is better than our competitors for the following reasons.

   a. We can plan paths not just around a small number of blocks, but around a large number of irregular viewsheds while simultaneously minimizing a realistic non-symmetric energy function.

   b. Our method works on high-resolution matrices. We could process cells with $3600 \times 3600$ points in 2006.

   c. We produce not just quasi-straight paths, but complicated curved paths.

3.4.8. Slope accuracy in lossy compression

We modified ODETLAP explicitly to incorporate slope into the objective function. That is, the compressed representation contains elevations which, when differenced, produce accurate slopes. The slope definition used is Zevenbergen-Thorne, as used by NGA.
3.4.9. Parallel ODETLAP

We implemented ODETLAP on a parallel computer, by partitioning the map into overlapping cells, parcelling them out to the processors, and merging the results. Details are in Stookey's thesis, included in a later section (appendix).

3.4.10. Evaluating hydrology preservation during compression
We present an error metric based on the potential energy of water flow to evaluate the quality of lossy terrain simplification algorithms.

We preserve hydrology better than ArcGIS.

4. Value of this project to DARPA

1. We make terrain data, and siting and path planning, more available.
2. Knowing where to site our observers helps us to shape the battlespace in our favor.
3. More compact data can be distributed to smaller computers in the field.
4. That helps our people to understand the world and reduces the fog of war.
5. This is an asymmetric advantage for our side.

5. Videos

The following short videos illustrating our work are on the web, in

The passwords are available from Dr Franklin.

1. **RPI-odetlap.wmv**

   This video demonstrates ODETLAP, showing how the reconstructed surface improves as more points are used.

2. **RPI-multipath.wmv**

   This video demonstrates computing many long paths around many forbidden zones (observers' viewsheds). That is a competitive advantage of our process, compared to some other methods.

3. **RPI-path-planning.mpg**

   This video follows a smuggler traveling along a surface while avoiding, to the greatest extent possible, the guards' viewsheds, and minimizing the path cost.

4. **RPI-Aug2008-Dan.m1v**

   This video shows ODETLAP being applied to the Ottawa LIDAR dataset, including the incremental point placement, and how the elevation and slope accuracies improve with the increasing number of points.

### 6. Software

The following software is on our web site here:


We wrote, or contracted for, all of it. The government may use it freely.

Each software suite is a gzipped tar file of a directory of files.

1. **tin.tz (1 MB)**

   This is our original Triangulated Irregular Network program, which is used
by ODETLAP. Altho written before this project, it is available here since we used it.

2. site.tz (11 MB)

This is our multi-observer siting program suite.

3. esri-siting.tz (16 MB)

This is the result of the subaward to Environmental Systems Research Institute, Inc. to develop a plugin to ArcGIS for multiple observer siting.

4. parallel-odetlap.tz (66 MB)

This is our parallel implementation of ODETLAP.

7. Press and Blog Mentions

This project was announced in the following stories and press releases.

7.1. Inside RPI

1 RPI Press release, 10/31/2005 (see appendix).
2 RPI School of Engineering Press Release
3 Rensselaer School of Engineering News
4 Improving Terrain Maps, Rensselaer Alumni Magazine Winter 2005-06

7.2. Outside RPI

1 Better terrain maps of Earth... and beyond, in Roland Piquepaille's Technology Trends, 5 nov 2005. (see appendix).
2 ZDNet, 11/5/2005 (see appendix).
3 Surf wax Government News
8. Publications / Public Presentations

The following publications and public presentations have resulted to date from this project. They are included in appendices.


5. **Terrain representation using tessellation of irregular planar tiles**. Metin Inanc and W Randolph Franklin, *16th Fall Workshop on Computational Geometry*, 10-11 Nov 2006, Smith College, Northampton MA, (poster presentation) [poster, extended abstract](#).

6. **Multiple observer siting on a compressed terrain**. Daniel Tracy, W Randolph Franklin and Franklin Luk, *16th Fall Workshop on*


10 **Smugglers and border guards - the GeoStar project at RPI.** W Randolph Franklin, Metin Inanc, Zhongyi Xie, Daniel M Tracy, Barbara Cutler, Marcus V A Andrade and Franklin Luk, *15th ACM International Symposium on Advances in Geographic Information Systems (ACM GIS 2007)*, Nov 2007, Seattle, WA, USA. **Talk** (much more recent than the paper.)

11 **Drainage network and watershed reconstruction on simplified terrain.** Jonathan Muckell, Marcus Andrade, W. Randolph Franklin, Barbara Cutler, Metin Inanc, Zhongyi Xie and Daniel M. Tracy. *17th Fall Workshop on Computational Geometry*, IBM TJ Watson Research Center, Hawthorne NY, 2-3 Nov 2007. **Poster, 2 page abstract, Video demoing Oahu dataset.**

12 **Approximating terrain with over-determined Laplacian PDEs.** Zhongyi Xie, Marcus A. Andrade, W. Randolph Franklin, Barbara Cutler, Metin Inanc, Daniel M. Tracy and Jonathan Muckell. *17th Fall Workshop on Computational Geometry*, IBM TJ Watson Research Center, Hawthorne NY, 2-3 Nov 2007. **Poster, 2 page abstract.**

13 **Slope accuracy and path planning on compressed terrain.** W.
Randolph Franklin, Daniel M. Tracy, Marcus Andrade, Jonathan Muckell, Metin Inanc, Zhongyi Xie and Barbara Cutler. \textit{13th International Symposium on Spatial Data Handling 2008 (SDH08)} June 2008, Montpelier, FR.

14 \textbf{Progressive transmission of lossily compressed terrain.} Zhongyi Xie, Marcus A. Andrade, W. Randolph Franklin, Barbara Cutler, Metin Inanc, Jonathan Muckell and Daniel M. Tracy. \textit{Conferencia Latinoamericana de Informática} (CLEI 2008), Santa Fe, Argentina. 8--12 Sep 2008. \texttt{paper}.

15 \textbf{Efficient viewed computation on terrain in external memory.} Marcus Andrade, Salles V. G. Magalhães, Mirella A. Magalhães, W. Randolph Franklin, Barbara M. Cutler, \textit{(conditionally accepted, Oct 2008)}.

16 \textbf{Path planning on complex terrain.} Dan Tracy, W Randolph Franklin, Barb Cutler, Franklin Luk, Marcus Andrade and Jared Stookey. \textit{18th Fall Workshop on Computational Geometry (FWCG 2008)}, Rensselaer Polytechnic Institute, Troy NY, 31 Oct - 1 Nov 2008. 2 page abstract, \texttt{talk}, \texttt{poster}.

17 \textbf{Operating on large geometric datasets.} W. Randolph Franklin. \textit{18th Fall Workshop on Computational Geometry (FWCG 2008)}, Rensselaer Polytechnic Institute, Troy NY, 31 Oct - 1 Nov 2008. 2 page abstract, \texttt{talk}.


20 \textbf{Evaluating hydrology preservation of simplified terrain}

---

**Student theses**


---

**9. Commercialization**

In 2007.2, two SBIRs hit the street that were clearly based on my research.

1. **A07-123 Novel Representations of Elevation Data.**

   That led to two phase I awards:
   1. W9132V-08-C-0012 Novel Representations of Elevation Data to Andrews Space, Inc.
   2. W9132V-08-C-0013 Novel Representations of Elevation Data to Numerica Corp.

2. **A07-126 Optimal Intervisibility Site Selection.** The solicitation cited me four times.
That led to the phase I award **W9132V-08-C-0005 Optimal Intervisibility Site Selection** to **Toyon Research Corp.**

(Unfortunately) I have no connection to any of those companies.

The fact that someone in the ARO considers this work important enough to issue solicitations to extend it says that DARPA is succeeding in having Geo* research transition to the Army.

### 10. Followon Projects

Our reputation, partly developed on this project, led to the following two projects.


2. Cutler, Franklin, and Tom Zimmie in RPI's Dept of Civil Engineering, received an NSF Cyber Enabled Discovery and Innovation award, *Fundamental Terrain Representations and Operations*, CMMI-0835762. That program had an expected 2% (1 in 50) funding rate, so we're happy. The purpose of this project is as follows.

The project, Fundamental Terrain Representations and Operations, unifies the fields of computational geometry, computer graphics, and civil engineering hydrology, resulting in a transformational ability to predict how erosion occurs, specifically in levee failure by overtopping, and, after a failure, to reverse-simulate what happened. This enables more efficient levee designs and so reduces the frequency and cost of levee failures in the United States. That will result in reduced flood damage in the US, which cost $50,000,000,000 in the 1990s, thereby saving money and lives. This project will continue and expand the PIs' strong track record of involving undergraduate students in research, hosting elementary and high school students and international visitors in their labs, and of involving women and other underrepresented groups. The vehicles
include the Research Experience for Undergrads and the Research Experience for Teachers programs, and RPI's own PREFACE program, which hosts high school student from underrepresented groups for two weeks in the summer. It will also expand continuing joint research with a Brazilian collaborator.

The project's goals are: a new representation for volumetric terrain, a.k.a. soil, that respects the geophysics of how surface water flows (hydrology) formed it, a better modeling of local erosion in terrain and earthen structures such as levees, an experimental plan for validation in Rensselaer Polytechnic Institute's geotechnical centrifuge, a predictive reverse simulation of earthen levee erosion, a visualization of non-homogeneous terrain erosion, an out-of-core parallel simulation on large terrain datasets, and a collaboration in Brazil. Therefore the accumulated data on past levee erosion will be transformed into knowledge about future erosion. One intellectually novel aspect of this research will be computational geometry representations of volumetric terrain in which the mathematics more closely aligns with the physics. It will not enforce a nonphysical continuity of elevation - cliffs are important in the real world. Even during progressive transmission, it will be hydrologically correct, with few or no interior local minima. Since erosion is not linear (gulleys don't cross each other and their elevations don't 'add'), such a representation cannot be linear, and so innovative and compute-intensive techniques are required.

Page last modified on March 09, 2009, at 10:28 PM
2 Press Releases

RPI 30
ZDnet 33
Roland Piquelay Technology Trends 36
Rensselaer Researcher Awarded DARPA Funding To Improve Terrain Maps

Troy, N.Y. — A Rensselaer researcher has been awarded $845,000 in federal funding to create improved computer representations of terrain on the surface of the Earth and beyond. The research could have a variety of both military and civilian applications, from strategically positioning soldiers to placing radio towers on the moon.

“I'm studying better ways to compress the massive amounts of terrain data now available from radar and laser scans of the Earth’s surface,” says W. Randolph Franklin, associate professor of electrical, computer, and systems engineering at Rensselaer Polytechnic Institute and principal investigator for the project, which is funded by the Defense Advanced Research Projects Agency (DARPA).

Current methods often produce unacceptable terrain maps, giving rise to errors that are clearly visible in any commercial mapping product, according to Franklin. For example, one common mapping software renders Niagara Falls as a gentle slope, while another has 50-foot elevation contours crossing a shoreline.

The program funding Franklin's work
— called Geo*, for GeoSpatial Representation and Analysis — exists because effective support for military operations requires better ways to represent Earth’s surface. A specific focus is on the need to improve navigation of unmanned aerial vehicles (UAVs).

“I will be researching and developing three different terrain representations,” Franklin says. “I will also be studying some important applications of terrain data.” One application is geared toward identifying the best sites to position a group of soldiers to allow them to see as much terrain as possible. Such a technology could also have civilian uses, such as in placing cell phone towers or locating visual nuisances where they would be the least visible.

“A far-out application for radio towers would occur when the moon or Mars are settled,” Franklin says. “Both have no ionosphere to enable long-distance radio, and the moon has no stable satellite orbits for potential communication satellites.” He suggests that ground-based radio relays, visible to each other, could be the best way to communicate on these surfaces.

DARPA is the central research and development organization for the Department of Defense (DOD). It manages and directs selected basic and applied research and development projects for DOD, and pursues research and technology where risk and payoff are both very high and where success may provide dramatic advances for traditional military roles and missions.

Published October 31, 2005
Contact: Jason Gorss
Phone: (518) 276-6098
E-mail: gorssj@rpi.edu
Better terrain maps of Earth... and beyond

Posted by Roland Piquepaille @ 11:34 am

The Defense Advanced Research Projects Agency (DARPA) thinks that today's computer maps of the Earth are inaccurate for its needs. So it recently awarded a grant to a Rensselaer Polytechnic Institute (RPI) researcher to develop better computer terrain maps of the surface of Earth and even our moon or Mars. RPI will develop new methods of compressing the data gathered by radar and laser scanning. Even if there is a specific focus to improve navigation of unmanned aerial vehicles (UAVs) and to position soldiers where they have the best visibility of their surroundings, other applications are envisioned, such as radio-rowers on the moon or Mars.

Here is what says W. Randolph Franklin, an associate professor of electrical, computer, and systems engineering at RPI.

"I'm studying better ways to compress the massive amounts of terrain data now available from radar and laser scans of the Earth’s surface," Franklin says.

Current methods often produce unacceptable terrain maps, giving rise to errors that are clearly visible in any commercial mapping product, according to Franklin. For example, one common mapping software renders Niagara Falls as a gentle slope, while another has 50-foot elevation contours crossing a shoreline.

And because DARPA has specific goals, such as improving the navigation of its UAVs, Franklin will develop several terrain maps.

"I will be researching and developing three different terrain representations," Franklin says. "I will also be studying some important applications of terrain data." One application is geared toward...
identifying the best sites to position a group of soldiers to allow them to see as much terrain as possible.

Below are two pictures illustrating other facets of the research work by W. Randolph Franklin (Credit: RPI/Franklin). On the top one, you can see "60 observers in the Adirondacks, viewed from above with a surface scan."

![Image of 60 observers viewed from above](image1)

And the second one shows "the same 60 observers, placed with a constraint of 'intervisibility,' so that each observer can see at least one other observer.

![Image of 60 observers with intervisibility constraint](image2)

These two pictures have been extracted from a previous work by Franklin and
his colleagues, "Multiple Observer Siting on Terrain With Intervisibility or Lo-Res Data." Here is a link to this paper from 2004 (PDF format, 6 pages, 816 KB).

And for more information about this Geospatial Representation and Analysis (GEO*) project at DARPA, here are two links to the original solicitation and to the project page.

Finally, here is Franklin's conclusion about future applications in space.

"A far-out application for radio towers would occur when the moon or Mars are settled," Franklin says. "Both have no ionosphere to enable long-distance radio, and the moon has no stable satellite orbits for potential communication satellites." He suggests that ground-based radio relays, visible to each other, could be the best way to communicate on these surfaces.

Sources: Rensselaer Polytechnic Institute news release, October 31, 2005; and various web sites

You'll find related stories by following the links below.

- Computers
- Geosciences
- Graphics
- Military Applications
- Space

Roland Piquepaille lives in Paris, France, and he spent most of his career in software, mainly for high performance computing and visualization companies. For disclosures on Roland's industry affiliations, click here.

Popular on CBS sites: March Madness | iPhone 3G | Fantasy Baseball | Antivirus Software | Recipes | Spore | Katy Perry

About CBS Interactive | Jobs | Advertise

© 2009 CBS Interactive Inc. All rights reserved. | Privacy Policy | Terms of Use
Better terrain maps of Earth... and beyond

The Defense Advanced Research Projects Agency (DARPA) thinks that today's computer maps of the Earth are inaccurate for its needs. So it recently awarded a grant to a Rensselaer Polytechnic Institute (RPI) researcher to develop better computer terrain maps of the surface of Earth and even our moon or Mars. RPI will develop new methods of compressing the data gathered by radar and laser scanning. Even if there is a specific focus to improve navigation of unmanned aerial vehicles (UAVs) and to position soldiers where they have the best visibility of their surroundings, other applications are envisioned, such as radio-rowers on the moon or Mars. Read more...

Sources: Rensselaer Polytechnic Institute news release, October 31, 2005; and various web sites

You'll find related stories by following the links below.

- Computers
- Geosciences
- Graphics
- Military Applications
- Space

8:38:43 PM Permalink
3 Papers and Public Presentations

Two novel surface representation techniques – Autocarto 2006 40
Tradeoffs when multiple observer siting on large terrain cells – SDH 2006 paper 51
Tradeoffs when multiple observer siting on large terrain cells – SDH 2006 talk 66
Compressing terrain datasets using segmentation – SPIE 2006 93
Terrain representation using tessellation of irregular planar tiles – FWCG 2006 poster 99
Terrain representation using tessellation of irregular planar tiles – FWCG 2006 abstract 100
Multiple observer siting on a compressed terrain – FWCG 2006 poster 102
Multiple observer siting on a compressed terrain – FWCG 2006 abstract 103
An improved LLL algorithm – Linear Alg & Apps 105
Surface compression using over-determined Laplacian approximation — SPIE 2007 117
Path planning on lossily compressed terrain — SPIE 2007 129
Smugglers and border guards – the GeoStar project at RPI – ACMGIS 2007 141
Smugglers and border guards – the GeoStar project at RPI – slides – ACMGIS 2007 149
Drainage network and watershed reconstruction on simplified terrain – FWCG 2007 abstract 167
Drainage network and watershed reconstruction on simplified terrain – FWCG 2007 poster 169
Approximating terrain with over-determined Laplacian PDEs – FWCG 2007 abstract 170
Approximating terrain with over-determined Laplacian PDEs – FWCG 2007 poster 172
Slope accuracy and path planning on compressed terrain — SDH 2008 173
Slope accuracy and path planning on compressed terrain — SDH 2008 talk 188
Progressive transmission of lossily compressed terrain — CLEI 2008 202
Efficient viewshed computation on terrain in external memory — 2008 212
Path planning on complex terrain – FWCG 2008 abstract 1 237
Path planning on complex terrain – FWCG 2008 talk 1 239
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path planning on complex terrain – FWCG 2008 poster</td>
<td>249</td>
</tr>
<tr>
<td>Operating on large geometric datasets – FWCG 2008 abstract</td>
<td>250</td>
</tr>
<tr>
<td>Operating on large geometric datasets – FWCG 2008 talk</td>
<td>252</td>
</tr>
<tr>
<td>Parallel ODETLAP for terrain compression and reconstruction — ACMGIS 2008 paper</td>
<td>265</td>
</tr>
<tr>
<td>Parallel ODETLAP for terrain compression and reconstruction — ACMGIS 2008 talk</td>
<td>274</td>
</tr>
<tr>
<td>Path planning on a compressed terrain — ACMGIS 2008 paper</td>
<td>290</td>
</tr>
<tr>
<td>Path planning on a compressed terrain — ACMGIS 2008 poster</td>
<td>294</td>
</tr>
<tr>
<td>Path planning on a compressed terrain — ACMGIS 2008 talk</td>
<td>295</td>
</tr>
<tr>
<td>Evaluating hydrology preservation of simplified terrain representations — ACMGIS 2008 poster</td>
<td>299</td>
</tr>
<tr>
<td>Evaluating hydrology preservation of simplified terrain representations — ACMGIS 2008 talk</td>
<td>300</td>
</tr>
</tbody>
</table>
Two Novel Surface Representation Techniques

W. Randolph Franklin, Metin Inanc, Zhongyi Xie
Rensselaer Polytechnic Institute, Troy, New York, 12180–3590, USA

Abstract

We present two new surface representation techniques: scooping and overdetermined Laplacian PDE approximation. Scooping reveals the terrain by removing material with a set of operators that resembling a 3-axis drill. Each operator has a square cross section, perhaps $7 \times 7$ posts, and a surface that is a polynomial of degree ranging from 0 to 3. Scoops may either partition the whole data cell, or else may be applied hierarchically and adaptively as needed to reduce errors. The longterm goal is for scoops to model geologic formation mechanisms such as water erosion.

The overdetermined PDE solves a overdetermined system of linear equations to produce a smooth surface approximation to a set of elevation posts. This representation has several advantages, such as the ability to infer local maxima inside nested rings of contours, and the ability to compute a best fit to an inconsistent set of inputs. The input data may be produced by an incremental Triangulated Irregular Network program, supplemented by an iterative insertion of the most inaccurately fitted points.

Both representations are part of the GeoStar project to lossily compress large terrain elevation matrices while preserving their usefulness for applications such as visibility and mobility.

1 Introduction

How shall terrain, meaning elevation above (or below) sea level, be represented in a computer? To simplify, we assume that elevation is single-valued; overhangs and caves are not considered here. However, we try to represent discontinuities, as they are perhaps the most important class of terrain feature. Discontinuities greatly affect visibility and mobility, and are easily recognizable.

The novel surface representation techniques presented here are intended to answer these questions: What terrain operators are appropriate, and how realistic they should be? There is a sweet spot: Fourier series are too unrealistic, but a complete geological evolution model is too complex.

2 Classical Terrain Representations

The following brief survey of three classical representations illustrates concerns and techniques that are relevant to our new representations, while presenting some new views.

---

*mail@wrfranklin.org, http://wrfranklin.org, +1 518 276 6077
†inancm@gmail.com
‡xiez@rpi.edu
2.1 Contour lines

Contour lines are perhaps best suited for hand sketching of a map. Since the actual terrain is not nearly as smooth as the contour line says, as typically realized on a paper map, they are quite lossy. This fact, combined with the elevation spacing between adjacent lines, means that important small features, such as gulleys and other minor elevation changes, may be unrepresentable. Therefore, determining visibility and mobility is much more error-prone. Also, representing very steep slopes and cliffs require distorting the contours.

Implementing contour lines on a computer reveals other disadvantages, arising from their essential nature as a hand-drawn technique. The first question is, how shall each line be represented? The obvious answer is as a polyline or sequence of points. If there are many points, then much space is required. If there are few points, then the spacing between lines is inaccurate, or the lines may even cross. The deep reason for this is the separate representation of each line, ignorant of their relationships.

One solution is to use level-set techniques, (Osher and Fedkiw, 2003), but that requires one of the techniques discussed later, so that the contours themselves become redundant.

An additional problem with contours is how to interpolate intermediate elevations, while avoiding problems like the following. When interpolating an elevation at some test point, the closest contour line in all 8 cardinal directions from the point may all be the same contour. This causes the test point, which lies between two contours, to have the elevation of one of the contours, which is probably wrong. Also, inferring a mountain top inside a set of nested contours is desirable. (Gousie and Franklin, 2005), (Gousie and Franklin, 2003), presents a new method that starts by computing new intermediate contours in between existing isolines. These are found by finding the shortest line segment that connects points on two neighboring contours with differing elevations. The midpoint of the line segment becomes a point on the intermediate contour. The contours are completed by connecting individual points. The new contours are then used as data for successive iterations, until an initial surface is formed. Peaks are computed by Hermite splines that follow the slope trend. Gaussian smoothing is applied to the entire surface or only to newly computed elevations, yielding an approximated or interpolated surface, respectively.

2.2 Triangulated Irregular Network

The Triangulated Irregular Network (TIN), a piecewise linear triangular spline, first implemented in cartography in 1973 by (Franklin, 1973), is purely a technique for computers; no one would implement a TIN by hand. After reviewing the TIN, this section discusses coding techniques for the TIN points. “Coding” a data representation means to represent its coordinates, pointers, or whatever, as a sequence of bits. It is often overlooked that an efficient coding is as important as the representation itself, and that a representation’s efficiency is not even a well formulated concept absent the coding.

One common implementation takes a set of data points \( \{(x, y, z)\} \), and uses a greedy point insertion to refine an initial triangulation into one with more points and a smaller error. A subtle issue is that inserting a point into the triangulation often increases the maximum error. However, in this case inserting the next point usually reduces the error considerably. The stopping criterion for this process can be the attainment of a desired maximum surface error, of the insertion of the desired number of points.
An alternate construction technique proceeds by constructing a complete triangulation of the
given data and then incrementally removing points. To our knowledge, these two techniques have
not been combined, although that would seem to reduce the error attained with a given number of
TIN points.

In either case, edges in the triangulation are flipped when necessary to main some property
such as Delauney. It is not \textit{a priori} obvious that that should be a more desirable property than
minimizing the total edge length or elevation error. However, our experiments find that Delauney
triangles do work better.

One misconception is that the unconstrained TIN does not adequately capture surface features
like ridge lines, which must instead be explicitly inserted into a constrained triangulation. However,
our experiments, on synthetic data constructed with both sharp and gradual, straight and curved,
ridges, and even discontinuities as would occur in a road cut, found no such problem. Perhaps this
was a problem with some other early implementations.

Another application of a TIN is a transform an irregular set of points into a regular grid, (Speck-
mann and Snoeyink, 1997). Here, the points are completely triangulated, and then grid points are
interpolated inside the triangles. This may be performed on very large datasets by sweeping up the
triangulation.

Implementing a TIN requires choosing an appropriate planar graph data structure. The obvious
answer, available in geometry packages, represents every topological dimension (point, edge, face),
and all their adjacency relations explicitly. The first problem is the storage cost required by all these
pointers, which can be an order of magnitude more than the elevations themselves. The second
problem is that keeping all this redundant information consistent as points are inserted and edges
flipped is tedious.

An advantage of requiring that the triangulation be Delauney is that storing any topology at
all is unnecessary, as it may be recomputed when needed. This is the endpoint of a sequence of
time–space tradeoffs. Succinct planar graph data structures, (Turan, 1984), which can store the
topology in a few bits per element, form an intermediate point in that tradeoff.

The TIN has the advantage of facilitating a \textit{progressive transmission} of the surface over a slow
communication link. That is, suppose that we wish to transmit the surface from server $S$ to client
$C$. $C$ may not even be certain that s/he wants the whole terrain until seeing a preview. Suppose that
$S$ computes a TIN using the insertion method, and then transmits the points one-by-one in their
insertion order. $C$ rebuilds the TIN by inserting the points as received. If the approximate surface
appears unsuitable, then $C$ tells $S$ to stop.

Once the TIN has been computed, the next step is to code it, or to represent it in as few bytes
as possible. This emulates other good data compression techniques, such as JPEG and BZIP,
whose space efficiency results from their design as a pipeline of compression techniques, with the
output from one step being the input to the next. For example, BZIP2 text compression contains
the following five steps in sequence: \begin{itemize}
\item Run length encoding
\item Burrows-Wheeler transformation
\item Move to front
\item Another run length encoding
\item Arithmetic encode.
\end{itemize}
JPEG image compression performs these steps in sequence: \begin{itemize}
\item Rotate RGB to YCrCb
\item Discrete cosine transform
\item Low-pass filter
\item Arithmetic encode.
\end{itemize}

To code the TIN points, we are considering various methods, but currently like the following
technique.

1. Start with the set of points, \{(x, y, z)\}, in the triangulation.
2. Compress the horizontal points, \( S_h = \{(x, y)\} \), separately, using one of the current bitmap compression techniques designed for facsimile transmission, (Salomon, 2000). A technique’s efficiency may be evaluated by comparing the size of its output to the information theoretic bound, obtained as follows.

Let \( M \) be the number of original potential points in the terrain. E.g., for a level-1 DEM, \( M = 1201^2 \).

Let \( N \) be the number of points in the triangulation.

Then, \( b \), the information content, in bits, or entropy, of \( S_h \) is

\[
\begin{align*}
  b &= \lg \left( \frac{M!}{N!(M-N)!} \right) \\
  b &\approx M \left( -p \lg p - q \lg q \right)
\end{align*}
\]

E.g., if \( N = 10^5 \), then \( p = 0.07, q = 0.93 \) and so \( b \approx 1.4 \cdot 10^5 \) bits, or 18 000 bytes. This is much less space than merely listing the \((x, y)\) coordinates.

One obvious potential optimization is to reduce \( M \), which is similar to reducing the number of significant digits in \( x \) and \( y \). The simplest realization is to subsample the input data before TINning it. However, reducing several points to one, say by averaging, loses perhaps too much information. A more sophisticated technique might proceed by first TINning the original data set, and then perturbing the selected TIN points so that they fall on a coarser grid. This has the advantage that the increased error is easily computable.

The definition of information content used here does assume that there is no structure to the points, that is, that there are no other usable relations between them. That is not quite true. In mountainous regions, the points will be close and in flat regions, widely spaced. However, it’s not clear either how much information content there is in this fact, nor how to exploit it.

3. Now the elevations need to be compressed. The order of the \( z \) is important since each \( z \) must be associated with the correct \((x, y)\). Without loss of generality, we can assume that the \((x, y)\) are lexicographically sorted. Then using a delta encoding for the \( z \) is reasonable, assuming that the consecutive elevations’ values are close. This property would become more true if we used a space filling curve instead of a mere lexicographic order for the \((x, y)\). That is not totally trivial for unevenly spaced points, but all that is necessary is that the ordering be unambiguously determinable from the set of points.

Various TIN extensions have been considered, such as using a higher degree triangular spline. That raises two issues. First, this idea’s effectiveness requires that the terrain generally have a higher degree continuity. More precisely, this requires that the terrain data being used possess this property. That distinction is relevant because terrain data is often artificially smooth. Second, there are technical difficulties with using higher degree triangular splines, compared to using Cartesian product splines.

There is an easy (but not as good) way and a hard (but better) way to use a higher degree triangular spline. The easy way goes as follows.

1. Compute a traditional TIN.
2. Fit a higher degree spline to this triangulation, while requiring the appropriate degree of continuity across each edge.

This method will work to the extent that the terrain data possesses the appropriate degree of continuity.

The hard but better way is to incrementally build up a higher degree spline. Note that determining the optimal spline points of any order is an exponential problem, so that some heuristic is necessary.

An advantage of TINs is that their resolution adapts to terrain regions of varying complexity. They are also not wedded to any particular coordinate system. However, any algorithm using them, such as visibility determination is complicated by the necessity of traversing the triangulation.

2.3 Gridded Elevation Matrices

If the TIN is too complicated, then a simple matrix, or array, of elevations is a reasonable alternative. The objection that this is not appropriate since the earth is not a developable surface, that is, cannot be flattened, is answered with the Riemannian manifold, (Jost, 2002). This is a geometric construct that represents a space of some dimension as an overlapping set of charts, each valid only in some specific limited region. The charts are organized in an atlas. The major application lies in modeling curved space-time.

An invalid objection to the matrix of elevations originates in the variable nature of terrain. Since a goal is to represent the terrain as compactly as possible, the matrix must be compressed, and good compression techniques adapt locally to the local information content of their input data.

Since so much effort has gone into image compression techniques, such as JPEG and SPIHT, it is worth trying them for terrain. SPIHT, (Said and Pearlman, 1993) lossily compresses elevation matrices quite well, (Franklin and Said, 1996).

In the following sections we propose other ways to compress elevation matrices.

3 New Terrain Representations

3.1 Scooping Operators

Scooping is an attempt to blast through the information theoretic limit for terrain compression by identifying and exploiting additional structure in the terrain. Such structure might originate from the geological formation processes.

Initially, we consider three different scoop operators to realize terrain scooping. The first one is flat and horizontal, the second one is flat and tilted and the third one is a quadratic equation. Those operators are used to scoop the terrain using either regular constant size tiles, or hierarchical quad-tree-like recursive tiles. Given the regular tiles’ limitations, such as the inability exactly to follow nonsquare features, the quality of their approximations is surprisingly good. We are already planning a future production system with a richer set of operators.
3.1.1 Horizontal (Degree 0) Scoops

The first operator is flat and horizontal. Given a tile of the terrain, this operator can approximate the terrain in three different ways. The first is way is to approach the tile from above and set the estimate to the maximum elevation in the tile. The second approximation is from below, thus we set the estimate to the minimum elevation in the tile. The third way is to set the estimate to the mean of all of the elevations in the tile. While both the overestimate and the underestimate of the terrain diverge from the mean with larger tile sizes, they also provide a convenient envelope, which may be of value to certain terrain applications. The range in the scoop given by the difference of the estimates is a local metric of terrain variance. All of these estimates are in effect bringing the terrain resolution down. The level of lowering resolution is controlled by the scoop size. They are convenient since they consist of a single parameter $c$, but are very simplistic as a model. The data model is $z = c$.

3.1.2 Planar (Degree 1) Scoops

The second terrain scooping operator is still flat as the first one but it is no longer required to be horizontal. In effect we introduce a second parameter, which is the normal vector, producing $z = ax + by + c$. We fit a plane to the tile can be done by finding the regression plane that minimizes RMS vertical error. Our test data included DTED Level 2 files, containing 3601 × 3601 elevations, such as the W111N31 cell shown in Figure 1. Experiments show that errors are rare and large errors are even rarer. Those occur on ridges and valley bottoms, where the planar model is not enough to capture data variance. Results from regularly tiled scoops of sizes: 3 × 3, 5 × 5 and 7 × 7 are shown in Figure 2. For instance, when using 7 × 7 scoops, over 90% of points had an absolute error under 4, or 0.05% of the elevation range.
<table>
<thead>
<tr>
<th>Degree of scoop surface</th>
<th>Planar</th>
<th>Quadratic</th>
<th>Cubic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of coefficients needed to represent one scoop</td>
<td>3</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>Number of scoops</td>
<td>301,859</td>
<td>159,626</td>
<td>95,099</td>
</tr>
<tr>
<td>Maximum absolute error</td>
<td>31</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>Mean absolute error</td>
<td>3.93</td>
<td>3.62</td>
<td>3.99</td>
</tr>
<tr>
<td>Number of scoops with error larger than 10</td>
<td>149</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Hierarchical scooping experiments on W111N31

### 3.2 Quadratic (Degree 2) Scoops

The third scooping operator is based on the quadratic equation \( z = ax^2 + by^2 + cxy + dx + ey + f \). Our hope is that the reduced number of required scoops will more than offset the doubled number of coefficients per scoop, especially when used in conjunction with the irregular size scooping algorithm described in the next section.

### 3.3 Hierarchical Scoops

This refinement is a recursive quadtree-like extension of any of the above methods, by varying the scoop size. Initially, the whole data cell is approximated with one scoop of the desired degree. If the maximum absolute error is larger than a threshold, which is 10 in this case, then we subdivide the cell into four subcells and repeat. This process stops at \( 3 \times 3 \) cells, which are represented by listing their 9 elevations.

Table 1 gives some results from testing linear through cubic hierarchical scoops on the W111N31 level-2 DTED cell, with 12.9 million points and elevation range \([809,2882]\). The number of scoops refers to how many scoops were needed to get either the maximum absolute error below the threshold of 10, or the scoop size down to \( 3 \times 3 \). Some of the latter scoops have a maximum absolute error over 10, as listed.

### 3.4 ODETLAP — Overdetermined Partial Differential Equations

This terrain representation technique is an extension of a Laplacian Partial Differential Equation (PDE). That interpolates from a set of data points to a complete array of elevations by defining an equation

\[
4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1}
\]

for every unknown non-border point. Border points are special cases, which are a little tricky to define properly. This equation has a physical origin. If the values represent temperatures in a planar medium instead of elevations, and the known points are places where heat is being applied or removed to maintain them at the given temperature, then the equation solves for the temperatures at all the unknown points. The Laplacian is easily solvable, say in Matlab, on arrays of several thousand square. However, the Laplacian has some limitations when used to interpolate terrain elevations, although these properties are physically correct for heat flow.

1. The interpolated values fall within the range of the known values, so local maxima, such as mountain tops, are never generated.
2. When a set of nested contours is interpolated, the generated surface droops, much like a cloth draped between two supports.

To remove those limitations, to provide other benefits, we extend this classical technique as follows.

1. Equation 1 is applied to every non-border point, known or unknown.

2. The known points also have a second equation,

\[ z_{ij} = h_{ij} \]  

where \( h_{ij} \) is the known elevation at that point, and \( z_{ij} \) is the computed elevation. Equation 2 is not trivial because our system of linear equations now has more equations than unknowns, that is, it is overdetermined, and almost certainly inconsistent. Therefore an exact solution is impossible. The best we can do is to solve the equations approximately, while minimizing the RMS errors. That is, if we combine the two types of equations into one matrix and one column vector: \( Az = b \) then the best we can do is to solve \( Az = b + e \) while minimizing the error \( e \). The solution is \( z = (A^T A)^{-1} A^T b \) although practical solution techniques use more efficient, albeit more complicated formulae. Although this system has very many unknowns, 1201\(^2\) for a level-1 DEM, most of the coefficients in \( A \) are zero, that is, the system is sparse. The key consideration when solving a sparse linear system is, to what extent are the zero entries filled in with nonzero values as the system is solved? Specialized solution techniques exist, and more are being developed. ODETLAP’s novelty is the overdetermined system, which was not feasible until recent large sparse system solution techniques were developed. Matlab can easily process cells with 400 × 400 posts (160000 unknowns). (Childs, 2003) processes larger systems.

There are several advantages to using an overdetermined Laplacian (ODETLAP) system of linear equations for approximating terrain. (Any resemblance to interpolation with springs is only superficial.) Approximation is now the correct term instead of interpolation since the fitted surface does not pass through the data points. Since the data is not exact, that is an advantage, as it leads to smoother surfaces while minimizing the error. ODETLAP can handle both continuous contour lines of elevations, which may have gaps, and isolated points, while producing a surface that infers mountain tops inside innermost contours while enforcing continuity of slope across contours and so showing no visible indication of the input contours, i.e., no generated terraces. So far as we know, no other interpolation method has all these advantages.

### 3.4.1 Choice of ODETLAP Input Points

ODETLAP approximates a surface to a set of points \( \{(x, y, z)\} \). How shall we select those points? We’ve tried several methods.

**Regular method:** Subsample every \( k \)-th point in both the \( x \) and \( y \) directions. This is easy.

**TIN method:** Run TIN on the input elevation file, and then use the first \( K \) points inserted into the TIN as the ODETLAP interpolation points.

**Refined method:** Iterate the ODETLAP process, as follows, by analogy to the TIN greedy insertion idea.
1. Start with some terrain, \( A \).
2. Pick \( S \), a set of input points, by the TIN method.
3. Run ODETLAP on them to reconstruct an approximate terrain, \( B \).
4. Compute the error between \( A \) and \( B \), and find \( M \), a set of points with greatest absolute error.
5. Insert the \( M \) points into \( S \).
6. Rerun ODETLAP.

### 3.4.2 Experiments

We experimented with those three methods on five sets of terrain with a resolution of \( 400 \times 400 \). Each method started with 1000 points; the refined method added 100 more worst points to the 1000 TIN points. The smoothness parameter \( R = 0.3 \), which means to value accuracy more than smoothness.

In the following table, “avg err is shorthand for “average absolute error” and “max err is shorthand for “maximum absolute error”.

<table>
<thead>
<tr>
<th>Data</th>
<th>TIN</th>
<th></th>
<th>Regular</th>
<th></th>
<th>Refined</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg err</td>
<td>max err</td>
<td>avg err</td>
<td>max err</td>
<td>avg err</td>
<td>max err</td>
</tr>
<tr>
<td>w113n3310</td>
<td>15.870</td>
<td>108.932</td>
<td>9.152</td>
<td>196.590</td>
<td>15.135</td>
<td>84.441</td>
</tr>
<tr>
<td>w113n3311</td>
<td>18.703</td>
<td>144.557</td>
<td>10.541</td>
<td>161.193</td>
<td>15.654</td>
<td>105.767</td>
</tr>
<tr>
<td>w113n3312</td>
<td>7.262</td>
<td>44.063</td>
<td>2.566</td>
<td>137.632</td>
<td>6.722</td>
<td>40.959</td>
</tr>
<tr>
<td>w113n3313</td>
<td>24.214</td>
<td>104.089</td>
<td>6.810</td>
<td>115.434</td>
<td>22.844</td>
<td>80.978</td>
</tr>
<tr>
<td>w113n3314</td>
<td>21.093</td>
<td>104.676</td>
<td>4.471</td>
<td>121.059</td>
<td>17.313</td>
<td>63.388</td>
</tr>
</tbody>
</table>

From the table, we have the following observations:

1. The regular method generally has a lower average error than the TIN method, but its maximum error is larger. The regular method’s errors are quite nonuniform, as is shown by its maximum error being much larger than its average error.

2. The TIN method generally has a much lower maximum error than the regular method, but a higher average error, so its errors are more evenly distributed, which is probably desirable.

3. The refined TIN method has the best maximum error. Its average error is less than the unrefined TIN method, but still larger than the regular error. Overall, this method has the most uniform error distribution, and we recommend it.

Because of the possibility that these results might be specific to the parameters used, we repeated the comparison of the (unrefined) TIN method to the refined TIN method for the 15 cases of \( R = 0.3, 1, 3 \) combined with \( N = 1000, 3000, 10000, 30000 \). The average error improved by an average of 5.2%, and the maximum error improved by an average of 11.2%.
3.4.3 Data Conflation

Sometimes we wish to supplement $A$, a large, low precision, terrain database with $B$, a small, high precision, database. $B$ covers only part of $A$, and is probably somewhat inconsistent with $A$ there. Since ODETLAP processes inconsistent equations, it can merge $A$ and $B$.

3.5 Correcting Errors in General

This is a general technique for refining the accuracy of any terrain representation. • Start with a terrain matrix $A$. • Apply any lossy technique to $A$, to produce $B$, which when uncompressed produces an approximate terrain $C$. • Compute the error matrix $E = A - C$. • Compress $E$, forming a representation $F$. • Store or transmit $(B, F)$. This method’s utility resides in the errors’ correlations, so that $|F|$ is small. Note that correlations in the errors mean that the original compression did not exploit all the structure in the terrain, but that’s another topic.

4 Future Work

Scooping was designed to be is analogous to scooping earth out of the side of a hill. Eventually, the scoops will follow a trajectory, starting from a given point, and proceeding in a downhill direction from there along a straight line trajectory for a given distance. It will scoop out a new gully of some width, whose bottom has some slope. As described, there are five parameters, although that could be varied. This will have several properties. • It will not create a local minimum. This desirable feature contrasts to every other known terrain representation method. • It naturally lends itself to the creation of complex drainage systems, again in contrast to other representations. • It will be quite nonlinear, and so has a power not available to linear methods.

Finally, any representation should have a more sophisticated evaluation criterion than absolute error. We are performing experiments on the effect the errors on important applications such as visibility and mobility, which may be combined into the smugglers’ path test. We site observers, compute viewsheds, and find an optimal path between some source and goal on the alternate terrain representation. Then, we compute the observers’ accurate viewsheds on the original elevation matrix, and count how much of that path, which was supposed to be completely hidden, is actually inside any of the accurate viewsheds. Preliminary results show that are representations are quite good under this metric.

5 Acknowledgements

This paper was supported by the National Science Foundation grant CCR 03-06502, and by DARPA/DSO under the Geo* program.

References


RPI Geo* Final Report 49/919 Autocarto 2006 Paper


Tradeoffs when Multiple Observer Siting on Large Terrain Cells

W. Randolph Franklin1 and Christian Vogt 2

1 Rensselaer Polytechnic Institute, Troy, New York, 12180–3590, USA, mail@wrfranklin.org
2 chvogt@gmail.com

Summary. This paper demonstrates a toolkit for multiple observer siting to maximize their joint viewshed, on high-resolution gridded terrains, up to 2402 × 2402, with the viewsheds’ radii of up to 1000. It shows that approximate (rather than exact) visibility indexes of observers are sufficient for siting multiple observers. It also shows that, when selecting potential observers, geographic dispersion is more important than maximum estimated visibility, and it quantifies this. Applications of optimal multiple observer siting include radio towers, terrain observation, and mitigation of environmental visual nuisances.

Key words: terrain visibility, viewshed, line of sight, siting, multiple observers, intervisibility

1 Introduction

Consider a terrain elevation database, and an observer, O. Define the viewshed as the terrain visible from O within some radius of interest, R, of O. The observer might be situated at a certain height, H, above ground level, and might also be looking for targets also at height H above the local ground. Also, define the visibility index of O as the fraction of the points within R of O that are visible from O. This paper goes beyond merely computing viewsheds of individual observers. It combines a fast viewshed algorithm with an approximate visibility index algorithm, to site multiple observers so as to jointly cover as much terrain as possible.

The multiple observers case is particularly interesting and complex, and has many applications. A cell phone provider wishes to install multiple towers so that at least one tower is visible (in a radio sense) from every place a customer’s cellphone might be. Here, the identities of the observers of highest visibility index are of more interest than their exact visibility indices, or than the visibility indices of all observers. One novel future application of siting radio transmitters will occur when the moon is settled. The moon has no ionosphere to reflect signals, and no stable satellite orbits. The choices for
long-range communication would seem to include either a lot of fiber optic cable or many relay towers. That solution is the multiple observer visibility problem.

As another example, a military planner needs to put observers so that there is nowhere to hide that is not visible from at least one. This leads to a corollary application, where the other side’s planner may want to analyze the first side’s observers to find places to hide. In this case, the problem is to optimize the targets’ locations, instead of the observers’.

Again, a planner for a scenic area may consider each place where a tourist might be to be an observer, and then want to locate ugly infrastructure, such as work yards, at relatively hidden sites. S/he may wish site a forest clearcut to be invisible to observers driving on a highway sited to give a good view. Finally, an architect may be trying to site a new house while following the planning board’s instruction that, “You can have a view, but you can’t be the view.”

Our programs may easily produce a set of observers with intervisibility, i.e., their views of each other form a connected graph, but we do not impose that constraint in the experiments reported here.

In contrast to many other researchers, we consider that speed of execution on large datasets is important. Many prototype implementations, demonstrated on small datasets, do not scale up well. That may happen either because of the size and complexity of the data structures used, or because of the asymptotic time behavior. For instance, even an execution time proportional to $N \log(N)$, where $N$ is the size of the input, is problematic for $N = 10^6$. In that case, the $\log(N)$ increases the time by a factor of 20. Some preliminary published algorithms may even be exponential if performing a naive search. Therefore, we strive for the best time possible.

In addition, large datasets may contain cases, which did not occur in the small test sets, that require tedious special programming by the designer. In a perfect software development process, all such cases would have been theoretically analyzed a priori, and treated. However, in the real world, testing on the largest available datasets increases our confidence in the program’s correctness.

Next, a large enough quantitative increase in execution speed leads to a qualitative increase in what we can do. Only if visibility can be computed efficiently, can it be used in a subroutine that is called many times, perhaps as part of a search, to optimize the number of observers. This becomes more important when a more realistic function is being optimized, such as the total cost. E.g., for radio towers, there may be a tradeoff between a few tall and expensive towers, and many short and cheap ones. Alternatively, certain tower locations may be more expensive because of the need to build a road. We may even wish to add redundancy so that every possible target is visible from at least two observers. In all these cases, where a massive search of the solution space is required, success depends on each query being as fast as possible.
Finally, although the size of available data is growing quickly, it is not necessarily true that available computing power is keeping pace. There is a military need to offload computations to small portable devices, such as a Personal Digital Assistant (PDA). A PDA’s computation power is limited by its battery, since, approximately, for a given silicon technology, each elemental computation consumes a fixed amount of energy. Batteries are not getting better very quickly; increasing the processor’s cycle speed just runs down the battery faster.

There is also a compounding effect between efficient time and efficient space. Smaller data structures fit into cache better, and so page less, which reduces time. The point of all this is that efficient software is at least as important now as ever.

The terrain data structure used here is either a $1201 \times 1201$ matrix of elevations, such as from a USGS level-1 Digital Elevation Model cell, or a $2402 \times 2402$ extract from the National Elevation Data Set. The relative advantages and disadvantages of this data structure versus a triangulation are well known, and still debated; the competition improves both alternatives. This current paper utilizes the simplicity of the elevation matrix, which leads to greater speed and small size, which allows larger data sets to be processed.

For distances much smaller than the earth’s radius, the terrain elevation array can be corrected for the earth’s curvature, as follows. For each target at a distance $D$ from the observer, subtract $D^2/(2E)$ from its elevation, where $E$ is the earth’s radius. The relative error of this approximation is $(D/(2E))^2$. It is sufficient to process any cell once, with an observer in the center. The correction need not changed for different observers in the cell, unless a neighboring cell is being adjoined. Therefore, since it can be easily corrected for in a preprocessing step, our visibility determination programs ignores the earth’s curvature.

The radius of interest, $R$, out to which we calculate visibility, has no relation to the distance to the horizon, but is determined by the technology used by the observer. E.g., if the observer is a radio communications transmitter, doubling $R$ causes the required transmitter power to quadruple. If the observer is a searchlight, then its required power is proportional to $R^4$.

In order to simplify the problem under study enough to make some progress, this work also ignores factors such as vegetation that need to be handled in the real world. The assumption is that it’s possible, and a better strategy, to incorporate them only later.

This paper extends the earlier visibility work in [9] and [11], which also survey the terrain visibility literature. The terrain siting problem was identified as far back as 1982 by Nagy, [2]. Other notable pioneer work on visibility includes [5, 18, 23]. [24] studied visibility, and provided the Lake Champlain W data used in this paper. [22] presented new algorithms and implementations of the visibility index, and devised the efficient viewshed algorithm that we use. One application of visibility is a more sophisticated evaluation of lossy compression methods, [1]. [3, 4, 19] analyze the effect of terrain errors on the
computed viewshed. [6] proposes modified definitions of visibility for certain applications. [17] explores several heuristics for siting multiple observers, and reports on the experimental tradeoffs that were observed. [25] discusses many line-of-sight issues. For more details on the results in this paper, see [13, 26]. An extended abstract was published in [7]. Lack of space here prevents the presentation of our experiments on the effect of lowered resolution on the quality of the siting.

The results reported here are part of a long project that may be called *Geospatial Mathematics*. Our aim is to understand and to represent the earth’s terrain elevation. Previous results have included these:

1. a Triangulated Irregular Network (TIN) program that can completely tin a 10801 × 10801 block of 3 × 3 level-2 DTEDs, [8, 10, 21],
2. Lossy and lossless compression of gridded elevation databases, [12], and
3. Interpolation from contours to an elevation grid, [15, 14, 16].

## 2 Siting Toolkit

This toolkit, whose purpose is to select a set of observers to cover a terrain cell, consists of four core C++ programs, supplemented with zsh shell scripts, Makefiles, and assorted auxiliary programs, all running in SuSE Linux. The impact of this toolkit resides in its efficient processing of large datasets.

1. **Vix** calculates approximate visibility indices of every point in a cell. **Vix** takes several user parameters: $R$, the radius of interest, $H$, the observer and target height, and $T$, a sample size. **Vix** reads an elevation cell. For each point in the cell in turn, **Vix** considers that point as an observer, picks $T$ random targets uniformly and independently randomly distributed within $R$ of the point, and computes what fraction are visible. That fraction is this point’s estimated visibility index.

2. **Findmax** selects a manageable subset, called the top observers, of the most visible tentative observers from **Vix**’s output. This is somewhat subtle since there may be a small region containing all points of very high visibility. A lake surrounded by mountains would be such a case. Since multiple close observers are redundant, we force the tentative observers to be spread out as follows.
   a) Divide the cell into smaller blocks of points. If necessary, first perturb the given block size so that all the blocks are the same size, ±1.
   b) In each block, find the $K$ points of highest approximate visibility index, for some reasonable $K$, e.g., 3. If there were more than $K$ points with equally high visibility index, then select $K$ at random, to prevent a bias towards selecting points all on one side of the block.

3. **Viewshed** finds the viewshed of a given observer at height $H$ out to radius, $R$. The procedure, which is an improvement over [11], goes as follows.
a) Define a square of side $2R$ centered on the observer.

b) Consider, in turn, each point around the perimeter of the square to be a target.

c) Run a sight line out from the observer to each target calculating which points adjacent to the line, along its length, are visible, while remembering that both the observer and target are probably above ground level.

d) If the target is outside the cell, because $R$ is large or the observer is close to the edge, then stop processing the sight line at the edge of the cell.

Various nastily subtle implementation details are omitted. The above procedure, due to [22], is an approximation, but so is representing the data as an elevation grid, and this method probably extracts most of the information inherent in the data. There are combinatorial concepts, such as Davenport-Schintzel sequences, which present asymptotic worst-case theoretical methods.

4. SITE takes a list of viewsheds and finds a quasi-minimal set that covers the terrain cell as thoroughly as possible. The method is a simple greedy algorithm. At each step, the new tentative observer whose viewshed will increase the cumulative viewshed by the largest area is included, as follows.

a) Calculate the viewshed, $V_i$, of each tentative observer $O_i$. $V_i$ is a bitmap.

b) Let $C$ be the cumulative viewshed, or set of points visible by at least one selected observer. Initially, $C$ is empty.

c) Repeat the following until it is not possible to increase $area(C)$, either because all the tentative observers have been included, or (more likely) because none of the unused tentative observers would increase $area(C)$.

i. For each $O_i$, calculate $area(C \cup V_i)$.

ii. Select the tentative observer that increases the cumulative area the most, and update $C$. Not all the tentative observers need be tested every time, since a tentative observer cannot add more area this time than it would have added last time, had it been selected. Indeed, suppose that the best new observer found so far in this step would add new area $A$. However we haven’t checked all the tentative new observers yet in this loop, so we continue. For each further tentative observer in this execution of the loop, if it would have added less than $A$ last time, then do not even try it this time.

In all the experiments described in the following sections, all the programs listed above are run in sequence. In each experiment, the parameters affecting one program are varied, and the results observed.
3 Vix and Findmax Experiments

Our goal here was to optimize VIX and FINDMAX, and to achieve a good balance between speed and quality. We used six test maps. Five of those maps were level-1 DEM maps, with 1201 × 1201 postings and a vertical resolution of 1 meter. The maps were chosen to represent different types of terrain, from flat planes to rough mountainous areas. Table 1 describes them, and Fig. 1 shows them.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mean</th>
<th>Min</th>
<th>Max</th>
<th>Range</th>
<th>St dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aberdeen east</td>
<td>420.5</td>
<td>379</td>
<td>683</td>
<td>304</td>
<td>36.5</td>
</tr>
<tr>
<td>Baker east</td>
<td>1260.9</td>
<td>546</td>
<td>2521</td>
<td>1975</td>
<td>376.9</td>
</tr>
<tr>
<td>Gadsden east</td>
<td>257.6</td>
<td>118</td>
<td>549</td>
<td>431</td>
<td>73.7</td>
</tr>
<tr>
<td>Hailey east</td>
<td>1974.1</td>
<td>954</td>
<td>3600</td>
<td>2646</td>
<td>516.3</td>
</tr>
<tr>
<td>Lake Champlain west</td>
<td>272.5</td>
<td>15</td>
<td>1591</td>
<td>1576</td>
<td>247.8</td>
</tr>
</tbody>
</table>

The sixth map is a National Elevation Data Set (NED) downloaded from the USGS “Seamless Data Distribution System”. From the original 7.5-minute map with bounds (41.2822, 42.4899), (−123.8700, −122.6882), the first 2402 rows and columns were extracted. This map is from a rough mountainous region, and was chosen to test our programs on a larger higher resolution map, since some siting programs might have difficulties here. Table 2 gives its statistics.

<table>
<thead>
<tr>
<th>Name</th>
<th>Mean</th>
<th>Min</th>
<th>Max</th>
<th>Range</th>
<th>St dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>California</td>
<td>706.9</td>
<td>205.9</td>
<td>2211.3</td>
<td>2005.4</td>
<td>2946.8</td>
</tr>
</tbody>
</table>

3.1 Testing Vix

These experiments tested the effect of varying $T$, the number of random targets used by Vix to estimate the visibility index of each observer. A higher $T$ produces more accurate estimates but takes longer. Note that precise estimates of visibility indexes are unnecessary since they are used only to produce an initial set of potential observers, called the top observers. Actual observers are selected from this set according to how much they increase the cumulative viewshed.

We performed these tests with various values of $R$ and $H$, on various datasets. The experiment consisted of five different test runs for all maps and an additional sixth test run for the larger map, as shown in Table 3. Each test
Tradeoffs when Multiple Observer Siting on Large Terrain Cells

run contained 10 different test cases, listed in Table 4. \( T = 0 \) gives a random selection of observers since all observers have an equal visibility index of zero.

**Table 3.** Parameter Values for the Different Test Runs of the Experiment (Italicized Case Only for the California Dataset)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius of interest ( R )</td>
<td>100 100 100 80 300 1000</td>
</tr>
<tr>
<td>Observer and target height ( H )</td>
<td>5 10 50 10 10 10</td>
</tr>
</tbody>
</table>

**Table 4.** Parameter Values for the Different Test Cases of the Experiment

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample size ( T )</td>
<td>0 2 5 8 12 15 20 30 50 200</td>
</tr>
</tbody>
</table>

Each test case was executed 20 times for the 1201 \( \times \) 1201 maps and 5 times for the 2402 \( \times \) 2402 map. Each time enough observers were selected to cover 80% of the terrain. (FindMax used a block size of 100 and 1008 top observers.) The mean number of observers over the 20 runs was reported.

Figure 2 shows results for \( R = 300 \) and \( H = 10 \). The results were normalized to make the output from the experiments with no random tests to
be 1. That is, 1 is the result that can be achieved by randomly choosing top observers for Site. Every value higher than one is worse than random, while every value lower than one is better. Figure 5 shows the Baker test case in more detail.

### 3.2 Testing FINDMAX

The purpose of the FINDMAX experiment was to evaluate the influence of FINDMAX on the final result of the siting observers problem. The two parameters evaluated were the number of top observers and the block size. The number of top observers specifies how many observers should be returned by FINDMAX. A larger number slows Site because there are more observers to choose from, but may lead to Site finally needing fewer observers. Therefore we want to keep this number as low as possible. It is computationally cheaper to increase the sample set in Vix than to increase the number of top observers. The block size specifies how much the top observers returned by FINDMAX are forced to spread out. A smaller number increases the number of blocks on a map and therefore reduces the number of top observers from a given block. This parameter has no influence on the computational speed.
Test procedure

The experiment for the number of top observers consisted of 9 different test cases. It was only conducted on the level-1 DEM maps. During the experiment the values for the number of top observers ranged from 576 to 10080. In all the test runs a block size of 100 was chosen, resulting in 144 blocks. 576 top observers produced 4 observers per block; 10080 top observers produced 70 observers per block. All different values for the number of top observers are given in Table 5, together with their resulting number of observers per block.

The experiment for the block size was different for level-1 DEM maps than for the larger map. In the case of the level-1 DEM maps there were 9 different test cases with values for block size ranging from 36 to 300. This resulted in having between 1 and 1089 blocks per map. The number of top observers was chosen to be 1000.

The actual number depends on the number of blocks since each block needs the same number of top observers. In case of the larger maps there are 8 different test cases with values for block size ranging from 80 to 2402. This results in having between 1 to 900 blocks per map. The number of top observers was chosen to be 2000. The actual number depends on the number
of blocks since each block needs the same number of top observers. All the different settings are given in Table 5.

**Evaluation**

In the sample size experiment, each test case was executed 20 times, with the entire application run each time until the site program was able to cover 80% of the terrain. Vix used $R = 100$, $H = 10$, and $T = 20$. The resulting number of observers needed to cover the 80% was noted, and the arithmetic mean from the results of the same test case calculated.

In the block size experiment, each test case was executed 20 times for the level-1 DEM maps and 5 times for the larger test. The evaluation of the results is slightly different. The site program ran until 100 (400 for the larger map) observers were sited. The parameters used for Vix were $R = 100$, $H = 10$, and $T = 20$. The amount of terrain visible by the final observers was then noted. The reason for changing the evaluation method was due to the problem that in some test cases we were not able to cover 80% of the cell.

Figure 3 shows for different maps how much terrain can be seen by 100 observers. For all data sets the parameters used were $R = 100$ and $H = 10$. The results are normalized by 1. For each map the best result achieved by any
Table 5. The parameters for block size and top observers are given for the different test cases. The values in the "Blocks" column represent the actual number of blocks used by Findmax given the size of the map and the parameters for block size and top observers. The values in the "obs/block" column represent the number of top observers that Findmax calculates for each block.

<table>
<thead>
<tr>
<th>Experiment Numbers</th>
<th>Top Observers</th>
<th>Block Size</th>
<th>100</th>
<th>100</th>
<th>100</th>
<th>100</th>
<th>100</th>
<th>100</th>
<th>100</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top Observers</td>
<td>576 864 1008 1296 1584 2016 3024 5040 10080</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Blocks</td>
<td>144 144 144 144 144 144 144 144 144</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Obs/Block</td>
<td>4 6 7 9 11 14 21 35 70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block Size</td>
<td>Top Observers</td>
<td>1089 1152 1083 1024 1125 1008 1024 1008 1008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Blocks</td>
<td>1089 576 361 256 225 144 64 36 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Obs/Block</td>
<td>1 2 3 4 5 7 16 28 63</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block Size</td>
<td>Top Observers</td>
<td>1089 1152 1083 1024 1125 1008 1024 1008 1008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Blocks</td>
<td>1089 576 361 256 225 144 64 36 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Obs/Block</td>
<td>1 2 3 4 5 7 16 28 63</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

value for the block size was considered to be 1. The results of the experiments using different values for the block size were scaled accordingly. Therefore the highest value that can be achieved is 1. Everything below one is worse.

Figure 4 shows for the larger map how much terrain can be seen by 100 observers. For the data sets the parameters used were $R = 100$ and $H = 10$. The results are normalized by 1. The best result achieved by any value for the block size was considered to be 1. The results of the experiments using different values for the block size were scaled accordingly. Therefore the highest value that can be achieved is 1. Everything below 1 is worse.

Figure 6 shows for different maps how many observers are needed to cover 80% of the data. For all data sets the parameters used were 100 for the radius of interest and 10 for the observer and target height. The results are normalized by 1. The results of the experiments that were achieved by computing 576 top observers was considered to be 1. Lower values are worse.

4 Conclusions

4.1 Vix Experiment

- A sample size of 20 to 30 random tests for Vix is a good balance between the quality of the result and the computational speed. Surprisingly this value is good for a wide range of parameters and terrain types.
- Vix improved the result on the level-1 DEM maps in the best case by reducing the amount of observers needed to 39% compared to randomly selecting top observers. The largest improvements were achieved for large
or rough terrain for large \( R \) or low \( H \). The smallest improvement was achieved on flat terrain.

- On the larger map the improvement of VIX was even bigger. Possible explanations are that this terrain is the roughest, and that there were fewer top observers per data point than in the smaller maps.

### 4.2 FINDMAX Experiment

- The block size should be chosen to be small, i.e., 2 to 5 observers per block. When covering a larger fraction of the terrain, more blocks with a smaller number of observers per block is important.

- Increasing the number of top observers in FINDMAX increases the quality of the result, but requires much more time. It is cheaper to increase the number of random tests in VIX, but there is a limitation for what can be achieved by increasing the number of random tests. The best results in the entire experiment were achieved with 10000 top observers. This might not be obvious when comparing the graph of the results from the VIX experiments with the results from the FINDMAX experiments. However, during the FINDMAX experiments a relatively large number of random tests was chosen. Therefore the visibility index for FINDMAX was of a high resolution.
The various tradeoffs mentioned above and the above experiments illuminate a great opportunity. They tell us that shortcuts are possible in siting observers, which will produce just as good results in much less time.

Another area for investigation is the connectivity of either the viewshed, or its complement. Indeed, it may be sufficient for us to divide the cell into many separated small hidden regions, which could be identified using the fast connected component program described in [20].

There is also the perennial question of how much information content there is in the output, since the input dataset is imprecise, and is sampled only at certain points. A most useful, but quite difficult, problem is to determine what, if anything, we know with certainty about the viewsheds and observers for some cell. For example, given a set of observers, are there some regions in the cell that we know are definitely visible, or definitely hidden? We have earlier demonstrated an example where the choice of interpolation algorithms for the elevation between adjacent posts affected the visibility of one half of all the targets in the cell.

![Effect of Varying the Number of Top Observers Returned by FINDMAX on the Number of Observers Needed to Cover 80% of the Cell, for Various 1201 × 1201 Cells](image-url)
This problem of inadequate data is also told by soldiers undergoing training in the field. Someone working with only maps of the training site will lose to someone with actual experience on the ground there.

Finally, the proper theoretical approach to this problem would start with a formal model of random terrain. Then we could at least start to ask questions about the number of observers theoretically needed, as a function of the parameters. Until that happens, continued experiments will be needed.

6 Acknowledgements

This paper was supported by the National Science Foundation grant CCR 03-06502, and by DARPA/DSO under the Geo* program.

References


Tradeoffs When Multiple Observer Siting on Large Terrain Cells
Spatial Data Handling (SDH)
Vienna, July 2006

W Randolph Franklin,
Christian Vogt
Rensselaer Polytechnic Institute
mail@wrfranklin.org, 703-447-7808
http://wrfranklin.org

Goals of this DARPA/DSO Geo* project

- Alternate terrain representations
- Compact; lossy - size / quality tradeoffs.
- Bias representations towards legal terrain
- Process datasets up to 10000x10000
- Uncompression speed is more important than compression speed.
- Evaluate on visibility, mobility metrics.
First: Study Terrain Properties

- Many local max, few local min
- Long range order - rivers
- Elevation and slope discontinuities are common, and are very important for mobility and visibility

Which is land, which water?

You can answer this => there is unexploited structure.
Where is This?

Answer
Examples of Other Errors

Fourier Series

- Widely used
- Excellent for representing many physical phenomena, like vibrations.
- Quite unsuitable for terrain.
- They assume $C^\infty$ continuity
- The truncated series is too smooth
- ...and has many local minima
Sample Results

- TIN: represented a 10800x10800 array to 3% max elevation error with 157,735 triangles.
- Scooping: represented w111n31 with 7x7 linear scoops with average error 0.1% and max error 2%.
- Using 7x7 scoops on one 3592x3592 dataset, multiobserver siting had only 6.5% error.
- ODETLAPping 400x400 piece of Lake Champlain W with 1/9 the points: error was 0.9m (0.1%).
- Combining TIN with ODETLAP: captures essence of surface with very few points.
- ODETLAP: Can fill radius 40 circles of missing data.

Test Data Complexities

- Varying Resolution
- Bunched Elevations
Testing Protocols

1. Elevation error: max, RMS
2. Visibility index: set of hi-vis observers.
3. Joint viewshed from multiple observer siting – are observers sited on alternate rep just as good?
4. Smugglers’ path planning – is path planned on alternate rep really hidden?

Interpolating LOS between posts

- Challenging
- Motivation: in one test, we tried various interpolation methods (min, max, linear)
- ½ of all the targets changed visibility
**The Known Unknowns of Viewsheds**

- Small changes in LOS interpolation cause large changes in visibility.
- One half of this cell has uncertain visibility.

---

**Protocol 3: Visibility Index Testing**

- Consider each post in term as an observer.
- Compute its visibility index.
  - Monte Carlo sampling: pick T random targets, compute their visibility, and report the fraction visible.
- Produce a map of all the visibility indexes.
- Compare the visibility index map of the original terrain representation to the map of the alternative representation.
Protocol 4: Multiple Observer Siting Testing

- Site a set of observers, $S_o$, on the original terrain rep.
- Site a set of observers, $S_a$, on the alternative terrain rep.
- Transfer $S_a$ to the original rep.
- Compare quality of $S_a$ to $S_o$.

Multiobserver Siting Steps

- Find approximate visibility index of every point in cell, using Monte Carlo sampling.
- Partition the cell into blocks and pick the best potential observers in each block.
- Using a greedy algorithm, select the best of the best observers.
- We have considerably studied the tradeoffs here.
Step 1: VIX – Approximate Visibility Indices

- For every potential observer in cell, pick T random targets within radius of interest.
- Run a line of sight to each target and see if visible.
- Estimated visibility index = fraction of targets that are visible.

Step 2: FINDMAX – Find Subset of Top Observers

- Goal: Reduce 3600x3600 posts to perhaps 1000 potential observer sites.
- Partition cell into blocks (to force observers to spread out).
- In each block, return observers with highest visibility indices.
Step 3: VIEWSHED – Find Top Observers’ Viewsheds

- Find (closest to) exact viewshed of every top observer from previous step.
- If radius of interest=200, then 200x200 bitmap.
- Run lines of sight from observer to perimeter, then back in and compute all visible points.
- Time: area of bitmap.

Step 4: SITE – Multiple Observer Siting

- Greedy selection of observers.
- At each step, pick observer whose viewshed adds most to cumulative viewshed.
- This is fast with bitmap operations.
- Selecting several hundred observers easy.
**Enforcing Intervisibility**

- After the first best-of-the-best, add only new observers that are inside the joint viewshed of the previous best-of-the-best.

![Image of viewshed comparison](image1.jpg)

**Effect of Intervisibility**

- This reduces the joint viewshed considerably.

![Graph of joint visibility index](image2.jpg)
Reduced Resolution Effect on Siting

- Lowering horizontal resolution lowers observer sitting quality.
- Lowering vertical resolution does not as much.
- Visibility, computed on lower resolution, is too high.

Experiments

Reducing horizontal resolution

Reducing vertical resolution

Visibility index

Resolution Reduction Factor
**Alternative Representations**

1. TIN
2. Scooping
3. ODETLAP
4. Combinations of the above, e.g., ODETLAP uses TIN points.

---

**Note: Good compression techniques are multistep**

**JPEG:**
1. Rotate RGB -> YCrCb
2. Discrete cosine transform
3. Low-pass filter
4. Arithmetic encode

**Text compression:**
1. Run length encoding
2. Burrows-Wheeler transformation
3. Move to front
4. Another run length encoding
5. Arithmetic encode
**TIN Status**

- We can process 10800x10800 arrays of posts in $\frac{1}{2}$ hr on PC.
- No external storage is used.
- Dataset formed by catenating nine 3601x3601 cells from data from the Savannah March kickoff meeting.
- Elevation range: 3600.

<table>
<thead>
<tr>
<th>Max elevation error</th>
<th>N output triangles</th>
<th>N out pts / N in pts</th>
<th>Exec time, CPU secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>157,735</td>
<td>0.0007</td>
<td>1292</td>
</tr>
<tr>
<td>10</td>
<td>5,320,089</td>
<td>0.023</td>
<td>1638</td>
</tr>
</tbody>
</table>

**TIN Features**

- Progressive resolution since points are inserted greedily.
- “Feature” points on peaks and ridgelines, and edges joining them, may be more important.
- Our TIN program selects them automatically; no need for manual identification and constrained triangulation.
- The points selected for the triangulation are assumed to be important, and can be fed into other methods, like ODETLAP.
- TIN is a piecewise linear triangular spline. Preliminary experiments with a higher degree spline showed no consistent improvement, and so were suspended.
**Alternative Representations**

1. TIN  
2. Scooping  
3. ODETLAP  
4. Combinations of the above, e.g., ODETLAP uses TIN points.

**Scooping Representations**

- This is longterm research.  
- The goal is to smash through the information theoretic barrier to terrain compression by utilizing geologic information.  
- We are pursuing several representations in parallel.
Scooping Status

Several subprojects:

- 3-axis milling machine experiments with set of simple drills.
- Complete cover test with parameterized sloped drills.
- Theoretical thinking about how scooping is different from, e.g., wavelets.

More General (Sloped) Drills

- Tradeoff powerful, large to encode, basis elements, vs small simple elements, of which we need more.
- Sweet point: basis elements resemble object being approximated.
- Purpose: to better understand scooping, while initiating experiments in slope-preservation during lossy compression.
- Underlying assumption: little long range correlation of elevation or slope.
**Regular Scoop Details**

- 7x7 Scoop size will represent 49 elevations using only 3 coefficients
- 7 is not a magic number but good enough for Level 2 DTED cells
- Large Errors are rare and mean error is very low, less than 2m
- Each scoop is a tilted plane which minimizes the error
- Regularity brings simplicity to the representation

---

**Regular 7x7 Tile Scoop Representation**

W111°N31° Reconstructed (Left), Error (Right)
Factor of 49 reduction in number of points
Percent of Elevation Errors on W 111° N 31°

7x7 Regular Sloped Scoop VIX Evaluation

- Comparing Postings with Visibility Index Larger Than 80%
- Original (Above), Reconstructed (Below)
- Yellow: High VIX
- Green: Low VIX
- Difference is not easy to discern
**Viewshed Evaluation of Regular Scooping**

- Dataset: 3595×3595
- Number of observers: 81
- Elevation range: 809 to 2882.
- Observer/target height is 10.
- Radius of interest: 300.

<table>
<thead>
<tr>
<th>Representation</th>
<th>%Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>tile3</td>
<td>1.31</td>
</tr>
<tr>
<td>tile5</td>
<td>3.67</td>
</tr>
<tr>
<td>tile7</td>
<td>6.52</td>
</tr>
<tr>
<td>tile75</td>
<td>5.62</td>
</tr>
<tr>
<td>tileavg75</td>
<td>3.16</td>
</tr>
</tbody>
</table>

**Multiobserver Viewshed Comparison**

![Original Terrain Rep](image1)
![Tile 7 Alternate Terrain Rep](image2)
**Alternative Representations**

1. TIN
2. Scooping
3. ODETLAP
4. Combinations of the above, e.g., ODETLAP uses TIN points.

**ODETLAP Review**

- Solve an overdetermined variant of a Laplacian PDE.
  - Known pts: \( z_{ij} = h_{ij} \)
  - All pts: \( 4z_{ij} = z_{i-1j} + z_{i+1j} + z_{ij-1} + z_{ij+1} \)
- Easily processes 400x400 arrays of elevation posts in Matlab.
**ODETLAP Advantages**

- Infers local maxima.
- Surface doesn’t droop.
- Utilizes isolated data, if available.
- Interpolates broken contours.
- Conformal (handles nested kidney-bean contours)
- Conflates inconsistent data, with user-defined weights.

**ODETLAP on Nested Squares**

- Various smoothness settings are possible.
- R=3 gives
  - Completely smooth silhouettes,
  - Average error = 2.7%
  - Max error = 12%.
**ODETLAP on Regular Points**

- Initially sample that with a subarray of regularly spaced points, every K points in each direction.
- When computing a complete surface from the sample points, parameter R trades off accuracy vs smoothness.
- Observe tradeoff of data size versus K, R on a mountainous region of the USGS Lake Champlain W level 1 DEM.

**Lk Champlain ODEPLAP Experiments**

<table>
<thead>
<tr>
<th>K</th>
<th>R</th>
<th>0.1</th>
<th>0.3</th>
<th>1.0</th>
<th>3.0</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>0.5</td>
<td>2.0</td>
<td>6.1</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.4</td>
<td>1.9</td>
<td>7.6</td>
<td>19</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.9</td>
<td>4.4</td>
<td>15</td>
<td>30</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.6</td>
<td>7.5</td>
<td>21</td>
<td>40</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2.4</td>
<td>11</td>
<td>28</td>
<td>49</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4.1</td>
<td>17</td>
<td>39</td>
<td>63</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>6.5</td>
<td>25</td>
<td>52</td>
<td>79</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>36</td>
<td>70</td>
<td>100</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>43</td>
<td>81</td>
<td>115</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>14</td>
<td>50</td>
<td>92</td>
<td>134</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>16</td>
<td>61</td>
<td>122</td>
<td>158</td>
<td>179</td>
<td></td>
</tr>
</tbody>
</table>

K: spacing of fitted points
R: smoothness vs accuracy
(Data range: 1378)
Factor of 9 Reduction

- ODETLAP used 1/9 as many points to represent this surface with an elevation error of 0.9m in a range of 1378.

Alternative Representations

1. TIN
2. Scooping
3. ODETLAP
4. Combinations of the above, e.g., ODETLAP uses TIN points.
**ODETLAPping Important Points**

- The preceding fits a surface to a regular grid of points.
- Fitting “important” points should be better.
- Use our TIN program, which, at each step, inserts the point farthest from the existing surface.
- Use the first N points selected by TIN.

**ODETLAPping TIN Points**

- Test: 400x400 sections: w11n3110, 3111, 3112. R=0.3
- Compare ODETLAPping first 1000 points selected by TIN with regular grid of 1000 points.
- Measure average, max. abs error over all original points.
- TIN: average is worse but max is better, but up to factor of 5.
- TIN points produce a better conditioned surface.
- Refined: Insert worst points into TIN ODETLAP. Result: even better conditioned surface.

<table>
<thead>
<tr>
<th>Data</th>
<th>TIN average error</th>
<th>Regular average error</th>
<th>Refined average error</th>
<th>TIN maximum error</th>
<th>Regular maximum error</th>
<th>Refined maximum error</th>
</tr>
</thead>
<tbody>
<tr>
<td>w11n3110</td>
<td>5.6</td>
<td>3.4</td>
<td>5.3</td>
<td>30.7</td>
<td>98.1</td>
<td>27.0</td>
</tr>
<tr>
<td>w11n3111</td>
<td>10.6</td>
<td>9.0</td>
<td>10.4</td>
<td>112.6</td>
<td>133.7</td>
<td>66.2</td>
</tr>
<tr>
<td>w11n3112</td>
<td>2.7</td>
<td>1.7</td>
<td>2.6</td>
<td>20.4</td>
<td>114.0</td>
<td>15.6</td>
</tr>
</tbody>
</table>
Fitting Regular vs TIN Points

- Original W111n3110 data (160,000 points)
- Fitting TIN points matches the character of the surface better

Fitting 100 Regular Points
Fitting 100 TIN Points

Fitting Regular vs TIN Points

- Original W111n3111 data
- Fitting TIN points matches the character of the surface better

Fitting 38 Regular Points
Fitting 30 TIN Points
W113n3311 - Regular vs Refined TIN

Themes

1. Nonlinearity is powerful.
2. Large memory is now available.
4. … But not always.
5. Respect the terrain.
6. Find fast heuristics.
7. Discover what we certainly know.
Smugglers' Path Planning on 16x Compressed "Scooped" Terrain Representation

Original 3595x3595
W119N31 Terrain: 12,924,025 d.f., Elev Range=2071

Compressed (7x7 Scoop): 791,267
d.f. (16x reduction), Mean abs error=1.7 (0.1%).

Compressed: Shortest Smugglers Path Computed Avoiding All 324 Viewsheds of Optimally Sited Observers

Evaluation: Optimal Path from Compressed Terrain Tested on Original Terrain Viewsheds – 14 of 4787 Points (0.3%) Are Errorneously Visible

Original: Joint Viewshed Computed for Same 324 Observers
Compressing Terrain Datasets Using Segmentation

W. Randolph Franklin, Metin Inanc
mail@wrfranklin.org, inancm@rpi.edu
Rensselaer Polytechnic Institute, 110 Eighth St Troy, NY, USA 12180

ABSTRACT

We propose a novel compression scheme to achieve lossy compression of elevation datasets. Our scheme does not use predictors in the traditional sense. Our predictors are based on planar dataset segments. We believe that this is a far better way of expressing context in an elevation dataset since it can capture continuities in different geometries and allows us to provide an error bound on the output.

Keywords: Terrain representation, compression

1. INTRODUCTION

Digital elevation data are more abundant than they have ever been but unfortunately the algorithms to efficiently process these data do not keep up with that. Current technologies allow for very fast elevation data acquisition. One such technology is LIDAR (Light Detection and Range), which is laser based range detector coupled with a GPS sensor. LIDAR allows for 20,000 to 50,000 readings per second. Each reading is stored as an xyz triplet where each coordinate is represented as an IEEE double amounting for 24 bytes per point. LIDAR was the technology used for the state of North Carolina after the Hurricane Floyd (1999) to map the whole state in the NC Floodplain Mapping Project [1]. Just the Neuse river basin (11% of the whole NC area) is approximately 500 million points and takes 11.2 GB to store. Another fast data acquisition technology is IFSAR (Interferometric Synthetic Aperture Radar). Shuttle Radar Topography Mission used interferometric radar technology to map 80% of the Earth’s landmasses, which amounts to approximately 120 million km² [2]. The amount of data collected in 11 days was in excess of 12 Terabytes. This is approaching the estimate for all the printed material in the Library of Congress. It has been estimated that if all the printed matter in the Library of Congress is stored as plaintext, it will take between 15 to 20 Terabytes to store.

In spite of all the inflation of the digital elevation data, the ways we handle it and store it have not advanced much. We still keep our data mostly as a grid of elevations and the compression standards used do not perform that well, which is not surprising as there is a few compression algorithms designed with terrain in mind. The gzip program (used to compress USGS DEMs) was originally designed to compress plaintext.

We propose a novel terrain representation method which organizes elevation postings in a small number of segments each associated with a plane equation. The list of plane equations is enumerated and each plane corresponds to a segment of elevation postings. This is an extension of our previous work which uses planar and high order patches to approximate the terrain data [3]. A typical segment would be a lake as the whole lake will perfectly fit a plane. This approach looks similar to the level set idea in [4] but is distinctly different as the planes can have wildly different inclinations. The segments are represented as a bitmap of plane identification numbers. The segment bitmap is further compressed using Huffman encoding.

Our main contribution is the novelty of the idea of using plane surfaces to capture elevation postings in segments, which are succinctly represented. Our method also has the advantage of bringing a user defined error bound on the restored data, which makes it applicable to terrain datasets. It is simple and extremely easy to implement as well. It is also open for future improvements.

In section two, we describe our algorithm and methodology. In section three, we present some results. In section four, we conclude discussing the future work. Section five contains the acknowledgements.

2. METHODOLOGY

Our input is a matrix of elevations. Each elevation posting is represented as a 16-bit integer. This is typical for many different elevation file formats like DEM and DTED. Our aim is to represent the elevation matrix in the most succinct
way possible. We want compression to preserve essential terrain properties like visibility and slope. One solution is to strive for a lossless compression, as the output will be the same as the input and we will recover every bit of the original. An example of lossless compression algorithm is implemented in JPEG2000, which uses reversible integer based wavelet transform to achieve this feat. A lossy compression scheme on the other hand will cause an output “similar” but not quite the same as the input. This type of scheme may be able to achieve compression ratios better than lossless algorithms at the price of some deterioration. However it is important to note that among the lossy compression algorithms designed for images there is not a good one applicable to terrains. The reason is that the similarity metric for images is not applicable in the domain of terrains. The human eye may be quite forgiving when discerning an image block compressed using DCT (Discrete Cosine Transform) from the original. However when applied on a terrain data the deviation from the original will be quite a lot. Among the lossy compression algorithms there are a few which retain terrain features and even those do not provide error bounds [5, 6]. All of the lossy image compression algorithms produce artifacts visible in 3D. Examples of problems are flattened patches, filled in river beds, flattened peaks and artificial drop-offs at the edge. While the lossy compression option of JPEG2000 is superior to the ordinary JPEG compression, it is recognized as unsuitable for terrain datasets, as it harms elevation processing [7, 8].

Most of the terrain datasets have established bounds of vertical error. JPEG compression using DCT will wreck havoc with the error bound and significantly change the original terrain. Thus a lossy compression algorithm must respect the error bound and increase it in a predictable way. If the vertical error bound is 16 meters, a lossy scheme which guarantees at most 5 meters of difference per elevation posting will be quite successful increasing the original error bound by only 5 meters.

Our approach is first to find the planes on which the elevation postings lie. Each plane will make a segment of the image. The strength of this approach lies in the fact that a segment may contain thousands of points effectively compressing a large number of points using just a few plane coefficients. The downside is that it is difficult to encode segments. We take on this problem further down in our discussion. A question that may arise is: why do we use planes? In a previous work [3], we show that due to locality properties terrains are very well represented using regular and irregular tiling of rectangular planes.

To produce the planes we use a basic kernel of size $2 \times 2$ which is moved over the elevation matrix. At each position the elevation postings falling within the kernel are used to calculate the best fitting plane using the ordinary regression, which provides an approximate solution (the four points of the kernel may not be coplanar) to the following matrix equation the with the unknown vector $(a, b, c)$.

$$
\begin{pmatrix}
  z_1 \\
  z_2 \\
  z_3 \\
  z_4
\end{pmatrix} =
\begin{pmatrix}
  x_1 & y_1 & 1 \\
  x_2 & y_2 & 1 \\
  x_3 & y_3 & 1 \\
  x_4 & y_4 & 1
\end{pmatrix}
\begin{pmatrix}
  a \\
  b \\
  c
\end{pmatrix}
$$

The $(x, y)$ coordinates are the coordinates of the grid crossings, thus they can be produced based on the horizontal and vertical offsets of the $2 \times 2$ kernel from the predefined origin. The $z$ values are the elevations themselves and they change as the $2 \times 2$ kernel position changes. The resulting plane has the following equation:

$$
z = ax + by + c
$$

This formula is a full blown plane equation, which gets rid of the $z$’s coefficient by normalizing it. It is interesting to note that the plane coefficients $a$ and $b$ usually have two or three significant decimal digits, while the coefficient $c$ can have up to 5 significant decimal digits. This is due to the fact that $a$ and $b$ are related to the direction of the plane’s normal, while $c$ is the constant factor which elevates the plane. A further work may exploit this property.

Upon computation of the plane equation of the kernel at a certain position we extend the plane over the whole grid of elevations. That is all $(x, y)$ coordinates of the elevation matrix are used to compute the $z$ values corresponding to the plane. Those elevation postings which are within a user defined distance (error) from the plane are included in the segment defined by the plane. The user defined distance parameter is the error bound guaranteed in the output. It is the vertical distance (along the $z$ coordinate) since the $x$ and $y$ coordinates are already fixed on the grid and cannot be a
source of error. An elevation posting can be in close proximity to more than one plane. In such a case the best idea is to assign it to a plane which has a larger membership set.

Using the 2x2 kernel exhaustively on the terrain data we find a large number of segments. A small subset of these segments is usually enough to cover the whole dataset. The problem is to find a subset of size as small as possible. There is an obvious brute-force algorithm which is unfortunately exponential since the number of possible subsets grows exponentially with the number of sets. We thus apply a greedy heuristic, which builds the subset in increments, always adding the set which will contribute the most to the subset. The end result is a subset of segments which covers the whole dataset.

**Greedy-Subset Heuristic**

```
Subset = empty
while Coverage(Set) < Whole Dataset
    maxCoverage = 0
    for i = available segments
        t = Set U i
        if Coverage(t) > maxCoverage
            maxCoverage = Coverage(t)
            bestSegment = i
        Subset = Subset U bestSegment
```

The subset produced by our heuristic is compact enough to allow compression. The segments are represented as a bitmap. If we happen to have 128 segments we can use a bitmap with a depth of 7 bits. So, not considering the plane coefficients we can achieve a compression ratio of 16:7. In reality we can do better. The number of elevation postings in the segments varies considerably. Thus we use Huffman coding to represent the segments in the bitmap. More popular segments will thus get shorter codes while less popular ones will have longer codes reducing the total size of the bitmap. But still this kind of representation has a lot of redundancy and we have not reached the potential of this representation.

The end result is a table of plane coefficients, which are stored as floating point numbers and a bitmap containing the segments. Together they can be used to reconstruct the terrain with a predictable error bound. If lossless operation is desired, it is possible to add the error matrix to the representation or decrease the user specified error bound to zero. However, both of these will inflate the resulting output.

3. **RESULTS**

We try our algorithm on six different datasets. All of our datasets contain 90,000 (300x300) elevation postings. Original size of each of the datasets is 180,000 bytes. The first three of the datasets are derived from the Adirondacks West USGS DEM file. They have 3 arc seconds of horizontal resolution, which corresponds approximately to 90m or 300 feet. The next three datasets are derived from the W 111° N 31° DTED file, which has a horizontal resolution of 30m or 100 feet. Table 1 contains the range of elevations in each of the datasets.

Table 1. Dataset Description.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Horizontal Resolution</th>
<th>Minimum Elevation</th>
<th>Maximum Elevation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset1</td>
<td>90m</td>
<td>278m</td>
<td>1591m</td>
</tr>
<tr>
<td>Dataset2</td>
<td>90m</td>
<td>122m</td>
<td>1076m</td>
</tr>
<tr>
<td>Dataset3</td>
<td>90m</td>
<td>29m</td>
<td>262m</td>
</tr>
<tr>
<td>Dataset4</td>
<td>30m</td>
<td>1097m</td>
<td>2014m</td>
</tr>
<tr>
<td>Dataset5</td>
<td>30m</td>
<td>1085m</td>
<td>1810m</td>
</tr>
<tr>
<td>Dataset6</td>
<td>30m</td>
<td>1398m</td>
<td>1627m</td>
</tr>
</tbody>
</table>
We use Matlab to implement the algorithm. We have not devised a file format for the output. We envision that if the system is ever deployed it will be easier to use a multi-file format not unlike the shape file format of ESRI. Such a multi-file format can have different files for the segment bitmap and plane coefficients.

We use error bounds of 5, 10, 20 and 40m for our experiments. In Table 2 we have the experiments with 5m of error bound. They exhibit the worst compression as they are most constrained and as a result have the largest number of segments to compress. As we relax the error bound in Tables 3 to 5 we get lower number of segments resulting in better compression ratios. The compression ratio is also affected by the dataset. Some datasets appear to compress better than others. This is caused by the fact that they can be represented with a very small number of segments, showing the low complexity of the datasets.

We plot the decreasing trend of the number of segments as the error bound increases in Figure 1.

Table 2. Algorithm details for error bound of 5m. The original size of each of the datasets is 180,000 bytes.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Segments (Subset Size) after the Heuristic</th>
<th>Size of the Huffman Compressed Segment Bitmap</th>
<th>Size of the Plane Coefficients Table</th>
<th>Total Size</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset1</td>
<td>830</td>
<td>88,225 bytes</td>
<td>9,960 bytes</td>
<td>98,185 bytes</td>
<td>1:0.5455</td>
</tr>
<tr>
<td>Dataset2</td>
<td>498</td>
<td>78,375 bytes</td>
<td>5,976 bytes</td>
<td>84,351 bytes</td>
<td>1:0.4684</td>
</tr>
<tr>
<td>Dataset3</td>
<td>72</td>
<td>25,074 bytes</td>
<td>864 bytes</td>
<td>25,938 bytes</td>
<td>1:0.1441</td>
</tr>
<tr>
<td>Dataset4</td>
<td>359</td>
<td>71,680 bytes</td>
<td>4,308 bytes</td>
<td>75,988 bytes</td>
<td>1:0.4222</td>
</tr>
<tr>
<td>Dataset5</td>
<td>203</td>
<td>61,474 bytes</td>
<td>2,436 bytes</td>
<td>63,910 bytes</td>
<td>1:0.3551</td>
</tr>
<tr>
<td>Dataset6</td>
<td>71</td>
<td>47,348 bytes</td>
<td>852 bytes</td>
<td>48,200 bytes</td>
<td>1:0.2678</td>
</tr>
</tbody>
</table>

Table 3. Algorithm details for error bound of 10m. The original size of each of the datasets is 180,000 bytes.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Segments (Subset Size) after the Heuristic</th>
<th>Size of the Huffman Compressed Segment Bitmap</th>
<th>Size of the Plane Coefficients Table</th>
<th>Total Size</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset1</td>
<td>435</td>
<td>76,450 bytes</td>
<td>5,220 bytes</td>
<td>81,670 bytes</td>
<td>1:0.4537</td>
</tr>
<tr>
<td>Dataset2</td>
<td>262</td>
<td>67,316 bytes</td>
<td>3,144 bytes</td>
<td>70,460 bytes</td>
<td>1:0.3914</td>
</tr>
<tr>
<td>Dataset3</td>
<td>26</td>
<td>17,576 bytes</td>
<td>312 bytes</td>
<td>17,888 bytes</td>
<td>1:0.0994</td>
</tr>
<tr>
<td>Dataset4</td>
<td>172</td>
<td>60,248 bytes</td>
<td>2,064 bytes</td>
<td>62,312 bytes</td>
<td>1:0.3462</td>
</tr>
<tr>
<td>Dataset5</td>
<td>101</td>
<td>51,987 bytes</td>
<td>1,212 bytes</td>
<td>53,199 bytes</td>
<td>1:0.2955</td>
</tr>
<tr>
<td>Dataset6</td>
<td>35</td>
<td>36,275 bytes</td>
<td>420 bytes</td>
<td>36,695 bytes</td>
<td>1:0.2039</td>
</tr>
</tbody>
</table>
Table 4. Algorithm details for error bound of 20m. The original size of each of the datasets is 180,000 bytes.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Segments (Subset Size) after the Heuristic</th>
<th>Size of the Huffman Compressed Segment Bitmap</th>
<th>Size of the Plane Coefficients Table</th>
<th>Total</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset1</td>
<td>206</td>
<td>64,464 bytes</td>
<td>2,472 bytes</td>
<td>66,936 bytes</td>
<td>1:0.3719</td>
</tr>
<tr>
<td>Dataset2</td>
<td>127</td>
<td>56,339 bytes</td>
<td>1,524 bytes</td>
<td>57,863 bytes</td>
<td>1:0.3215</td>
</tr>
<tr>
<td>Dataset3</td>
<td>16</td>
<td>13,725 bytes</td>
<td>192 bytes</td>
<td>13,917 bytes</td>
<td>1:0.0773</td>
</tr>
<tr>
<td>Dataset4</td>
<td>82</td>
<td>48,364 bytes</td>
<td>984 bytes</td>
<td>49,348 bytes</td>
<td>1:0.2742</td>
</tr>
<tr>
<td>Dataset5</td>
<td>53</td>
<td>40,136 bytes</td>
<td>636 bytes</td>
<td>40,772 bytes</td>
<td>1:0.2265</td>
</tr>
<tr>
<td>Dataset6</td>
<td>20</td>
<td>25,617 bytes</td>
<td>240 bytes</td>
<td>25,857 bytes</td>
<td>1:0.1436</td>
</tr>
</tbody>
</table>

Table 5. Algorithm details for error bound of 40m. The original size of each of the datasets is 180,000 bytes.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Segments (Subset Size) after the Heuristic</th>
<th>Size of the Huffman Compressed Segment Bitmap</th>
<th>Size of the Plane Coefficients Table</th>
<th>Total</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset1</td>
<td>100</td>
<td>52,487 bytes</td>
<td>1,200 bytes</td>
<td>53,687 bytes</td>
<td>1:0.2983</td>
</tr>
<tr>
<td>Dataset2</td>
<td>58</td>
<td>42,722 bytes</td>
<td>696 bytes</td>
<td>43,418 bytes</td>
<td>1:0.2412</td>
</tr>
<tr>
<td>Dataset3</td>
<td>6</td>
<td>11,696 bytes</td>
<td>72 bytes</td>
<td>11,768 bytes</td>
<td>1:0.0654</td>
</tr>
<tr>
<td>Dataset4</td>
<td>47</td>
<td>36,043 bytes</td>
<td>564 bytes</td>
<td>36,607 bytes</td>
<td>1:0.2034</td>
</tr>
<tr>
<td>Dataset5</td>
<td>25</td>
<td>26,513 bytes</td>
<td>300 bytes</td>
<td>26,813 bytes</td>
<td>1:0.1490</td>
</tr>
<tr>
<td>Dataset6</td>
<td>11</td>
<td>13,186 bytes</td>
<td>132 bytes</td>
<td>13,318 bytes</td>
<td>1:0.0740</td>
</tr>
</tbody>
</table>

Fig. 1. With the increase in the error bound from 5m to 40m the number of segments decreases.
4. CONCLUSION AND FUTURE WORK

Our algorithm is far from having reached its potential. Yet it is simple and robust providing good compression ratios. It derives its strength from its unique approach to the problem of representing the terrain data. The concept is that the terrain data has inherent properties, which provide for a structure we try to exploit.

The algorithm presented has several areas for improvement. First we would like to experiment with different heuristics, which may provide us with smaller subsets of segments. Considering the size of the solution space, the exact solution algorithm may not be polynomial but an approximation can be found. Another part of the algorithm which has room for improvements is the representation of the segments. Compressing the segment bitmap using a different encoder or representing segments in a different way will also help improve the performance. An idea is to try to get rid of the segment bitmap and use plane intersections instead. Furthermore, in the current scheme we do not compress plane coefficients. The fact that the plane coefficients have different number of significant digits can also be used to improve compression.

5. ACKNOWLEDGEMENTS

This paper was supported by the National Science Foundation grant CCR 03-06502, and by DARPA/DSO under the Geo* program. We thank Prof. Frank Luk, Prof. Barbara Cutler, Prof. Caroline Westort, Zhongyi Xie and Dan Tracy for valuable discussions on terrain representation.

REFERENCES

Terrain Representation Using Tessellation of Irregular Planar Tiles

Motivation
Terrain elevation datasets take up a lot of space. We are looking at the problem of compacting these datasets. Our approach consists of clustering elevations close to a plane and representing these in a single shot.

The Algorithm
1. Partition the terrain using a regular square tessellation with each square being 2x2.
2. Fit the elevation data on each square to the best (least-squares) plane.
3. Each plane is evaluated over the whole terrain. The region where the plane is within X meters from the original makes an irregular tile.
4. From the very large set of irregular tiles pick the smallest possible subset which covers the whole terrain.
5. Transmit all of the irregular tiles. Each irregular tile is represented by its location on the terrain grid and the plane on it.

Data Fitting
Multiple Linear Regression is used to fit the four elevations within each of the square tiles to a plane.

\[
\begin{bmatrix}
x_1 & y_1 & 1 & 1 \\
x_2 & y_2 & 1 & 1 \\
x_3 & y_3 & 1 & 1 \\
x_4 & y_4 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
\alpha \\
\beta \\
\gamma \\
\delta \\
\end{bmatrix}
= \begin{bmatrix}
z_1 \\
z_2 \\
z_3 \\
z_4 \\
\end{bmatrix}
\]

Plane Evaluation
The plane is evaluated over the whole terrain subject to an error guarantee.

\[
\overline{z}(x,y) = \alpha x + \beta y + \gamma \\
\forall (x,y) \quad |z(x,y) - \overline{z}(x,y)| \leq \text{Max Error}
\]

Results for 400x400 Elevation Datasets
There is a drastic decrease in the number of tiles with the decrease in the error bound.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Res (m)</th>
<th>Min (m)</th>
<th>Max (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 1</td>
<td>90</td>
<td>127</td>
<td>1591</td>
</tr>
<tr>
<td>Set 2</td>
<td>60</td>
<td>225</td>
<td>707</td>
</tr>
<tr>
<td>Set 3</td>
<td>45</td>
<td>29</td>
<td>262</td>
</tr>
<tr>
<td>Set 4</td>
<td>30</td>
<td>1097</td>
<td>2014</td>
</tr>
<tr>
<td>Set 5</td>
<td>15</td>
<td>1085</td>
<td>1610</td>
</tr>
<tr>
<td>Set 6</td>
<td>10</td>
<td>1396</td>
<td>1627</td>
</tr>
</tbody>
</table>

References

16th Fall Workshop on Computational Geometry, Nov 10-11, 2006, Smith College
Terrain Representation Using Tessellation of Irregular Planar Tiles

Metin Inanc, W. Randolph Franklin
inancm@rpi.edu, mail@wrfranklin.org
Rensselaer Polytechnic Institute,
110 Eighth St, Troy, NY 12180

Executive Summary
A novel terrain representation based on tessellation of irregular planar tiles is being developed. We start with a regular square tessellation. Each square tile is used as a generator for a potential irregular planar tile. From all of the potential irregular tiles those which contribute most to the terrain coverage are picked and used for the tessellation.

Introduction
The algorithm developed can be used to represent digital terrain elevation maps with certain quality guarantees. To achieve this goal a representation based on irregularly shaped planar tiles is being used. We use an objective function to evaluate the fitness of the planar tiles used in the tessellation. The quality guarantee is a parameter of the objective function and bounds the maximum absolute error of the representation. The maximum absolute error is defined as the maximum absolute difference between the representation and the original terrain.

Methodology
We start by building a regular square tessellation of the original terrain. The typical size of a square tile is 2x2. Each of the square tiles contains 4 elevation values. Those are modeled using the following linear model:

\[
\begin{pmatrix}
z_1 \\
z_2 \\
z_3 \\
z_4
\end{pmatrix} =
\begin{pmatrix}
x_1 & y_1 & 1 & a \\
x_2 & y_2 & 1 & b \\
x_3 & y_3 & 1 & c \\
x_4 & y_4 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
\varepsilon_1 \\
\varepsilon_2 \\
\varepsilon_3 \\
\varepsilon_4
\end{pmatrix}
\]

While \(x\) and \(y\) are the given variables, \(z\) models the predicted variable; the elevation. Multiple linear regression is used to solve for the model parameters: \(a\), \(b\) and \(c\). The resulting parameters are the coefficients of the best fitting (in the least squares sense) plane equation. The plane equation \(\tilde{z}(x, y) = ax + by + c\) is evaluated over the whole terrain using our objective function:

\[\forall (x, y) \quad z(x, y) - \tilde{z}(x, y) \leq \text{Max Error Guarantee}\]

The \((x, y)\) tuples which satisfy the objective function make up a potential irregular planar tile. Each 2x2 square results in a potential planar tile.

In the next step the set of potential planar tiles is subjected to the picking procedure, which selects a subset of planar tiles to build a tessellation. Since the solution space is exponential, in a set with \(N\) members there are \(2^N\) possible subsets, we use a greedy heuristic which usually gives an excellent solution:

Greedy-Subset Heuristic
Subset = empty
while Coverage (Subset) < Whole Terrain
maxCoverage = 0
for i = available Tiles
\[
t = \text{Subset} \cup i \\
\text{if Coverage} (t) > \text{maxCoverage} \\
\text{maxCoverage} = \text{Coverage} (t) \\
\text{bestTile} = i \\
\text{Subset} = \text{Subset} \cup \text{bestTile}
\]

The result of the heuristic is a subset of irregular planar tiles, which tessellate the terrain subject to our objective function.

**Data Structures**

We use an indexed bitmap to keep the irregular tessellation. We have tried modern image compression methods like lossless JPEG 2000 on the resulting bitmaps without success. Thus we implement Huffman compression which assigns shorter codes for the larger tiles in the tessellation. We keep the planar coefficients of the irregular tiles in an indexed table and we have devised a method which aggressively compresses them with very little error on the resulting terrain.

**Results**

We use six different exemplary datasets all containing 300x300 elevations with varying terrain complexity. The first three of the datasets have horizontal resolution of approximately 90 m; the last three have a resolution of 30 m. The number of tiles drops exponentially with the increase in the error bound.

<table>
<thead>
<tr>
<th>Resol Elevation</th>
<th>Min Elevation</th>
<th>Max Elevation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dset1 90m</td>
<td>278m</td>
<td>1591m</td>
</tr>
<tr>
<td>Dset2 90m</td>
<td>122m</td>
<td>1076m</td>
</tr>
<tr>
<td>Dset3 90m</td>
<td>29m</td>
<td>262m</td>
</tr>
<tr>
<td>Dset4 30m</td>
<td>1097m</td>
<td>2014m</td>
</tr>
<tr>
<td>Dset5 30m</td>
<td>1085m</td>
<td>1810m</td>
</tr>
<tr>
<td>Dset6 30m</td>
<td>1398m</td>
<td>1627m</td>
</tr>
</tbody>
</table>

**Conclusion**

We have a novel method of terrain representation, which extends our previous work in [1], where a method based on regular tile tessellation is presented. The same problem of lossy terrain representation was studied in [2] using a very different perspective. Our method is very different from the level set method approach which is based on contour lines [3].

**Acknowledgements**

This paper was supported by the National Science Foundation grant CCR 03-06502.

**References**

Multiple Observer Siting
Site a group of observers so as to maximize the amount of visible terrain, e.g.: place cell phone towers in order to optimize the coverage area.

Use multiple observer siting to evaluate the quality of the compression.

Terrain Compression Techniques
1. JPEG 2000
2. TIN+ODETLAP (Overdetermined Laplacian Solver)

Perform multiple-observer siting on the alternate representation to generate a set observers, along with the corresponding joint viewedsh.

Path Planning
Smuggler’s Path: Compute the shortest path between opposite corners while avoiding the visible areas. Compare the lengths of the paths produced on the original and alternate representations of the terrain. Also, examine how much of the path computed on the alternate terrain is visible on the original.

<table>
<thead>
<tr>
<th></th>
<th>JPEG 2000</th>
<th>TIN + ODETLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Viewshed Size</td>
<td>123416</td>
<td>125520</td>
</tr>
<tr>
<td>Alternate Viewshed Size</td>
<td>135998</td>
<td>137117</td>
</tr>
<tr>
<td>Viewshed Error</td>
<td>9.76%</td>
<td>9.04%</td>
</tr>
<tr>
<td>Path Length Error</td>
<td>5.81%</td>
<td>0.23%</td>
</tr>
<tr>
<td>Path Visibility Error</td>
<td>1.56%</td>
<td>0.27%</td>
</tr>
</tbody>
</table>

Red: High elevation
Blue: Low elevation
Bright: Visible
Dark: Hidden

The straight-forward implementation of A* to compute the shortest path results in only the Manhattan distance being minimized. We propose a new scheme which seeks to minimize the Euclidean distance, without sacrificing efficiency.
Multiple Observer Siting on a Compressed Terrain

D. M. Tracy, W. R. Franklin, and F. T. Luk
Rensselaer Polytechnic Institute

1. Introduction

We consider the problem of multiple observer siting on a compressed terrain. The problem is to site a group of observers so as to maximize the amount of visible terrain; an application is the placement of watchtowers to observe a territory. In real life, we have access to only a compressed terrain because the original terrain requires too much storage. For our problem, we want to evaluate the quality of the compression provided by the compression scheme.

The usual way to evaluate a compression scheme is to calculate the average and the maximum errors of the compressed terrain. The approach may not be appropriate for this application because we are interested in avoiding detection. In this paper, we propose a new test protocol and new error metrics. Our new protocol is to compute a minimum-length path from the northwest corner of the terrain to the southeast corner, while avoiding detection, and our new error metrics are to examine path lengths and visibility errors.

2. Algorithm

We apply an algorithm due to Franklin and Vogt [1] to do the multiple observer siting. Our algorithm consists of four steps. First, compress the original terrain and then uncompress it to generate the alternate representation. Second, perform multiple-observer siting [1] on the alternate representation to generate a set of observers, along with the corresponding joint viewshed. Third, site the same group of observers on the original terrain, and compute the new joint viewshed. Finally, compare the two viewsheds to determine the difference in visibility between the two representations.

Our new test protocol for path finding is also a four step algorithm. Indeed, the first three steps are identical to the corresponding first three steps above. In the fourth step, we apply the A* algorithm to find the paths for both the original and the alternate representations. A word of caution in step four: the naïve method for computing the shortest path results in only the Manhattan distance being minimized. We propose a new scheme which seeks to minimize the Euclidean distance, without sacrificing efficiency. Our new error metrics consist of comparing the two path lengths and determining how much the paths are visible on the original representation.
3. Results

Using our new protocol and error metrics, we compare two compression schemes: the famous JPEG 2000 and our new approach of TIN+ODETLAP (Overdetermined Laplacian Solver). The two schemes are competitive. There are cases where JPEG 2000 performs better, and there are cases where TIN+ODETLAP wins out. It appears that our new approach is better when the terrain is very heterogeneous, as illustrated in the example below.

<table>
<thead>
<tr>
<th></th>
<th>JPEG 2000</th>
<th>TIN + ODETLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Viewshed Size</td>
<td>123416</td>
<td>125520</td>
</tr>
<tr>
<td>Alternate Viewshed Size</td>
<td>135998</td>
<td>137117</td>
</tr>
<tr>
<td>Viewshed Error</td>
<td>9.76%</td>
<td>9.04%</td>
</tr>
<tr>
<td>Path Length Error</td>
<td>5.81%</td>
<td>0.23%</td>
</tr>
<tr>
<td>Path Visibility Error</td>
<td>1.56%</td>
<td>0.27%</td>
</tr>
</tbody>
</table>

References


An improved LLL algorithm

Franklin T. Luk*, Daniel M. Tracy 1

Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

Received 18 January 2007; accepted 27 February 2007

Submitted by M.K. Ng

Abstract

The LLL algorithm has received a lot of attention as an effective numerical tool for preconditioning an integer least squares problem. However, the workings of the algorithm are not well understood. In this paper, we present a new way to look at the LLL reduction, which leads to a new implementation method that performs better than the original LLL scheme.

© 2007 Published by Elsevier Inc.

Keywords: LLL algorithm; Integer least squares; Unimodular transformation; Reduced basis; Gauss transformation; QR decomposition; Plane reflection; Condition number

1. Introduction

The famous algorithm due to Lenstra, Lenstra and Lovasz (LLL [4]) has many important applications; for example, wireless communication, cryptography, and GPS (see [3] and references therein). In some of these applications, researchers use the LLL algorithm as a preconditioner in the solution of an integer least squares problem. Although the LLL algorithm is often referred to as an integer Gram–Schmidt procedure, no one has fully analyzed its numerical behavior. In this paper, we present a new way to examine the LLL reduction. Our idea leads to a new, generalized LLL technique that uses orthogonal instead of Gauss transformations in the reduction process.

Our paper is organized as follows. In Section 2, we describe the problem of integer least squares. We present the idea of a reduced basis and the LLL algorithm in Sections 3 and 4, respectively.

* Corresponding author.
E-mail address: luk@cs.rpi.edu (F.T. Luk).

1 This work was supported in part by DARPA/DSO under the Geo* program.

0024-3795/$ - see front matter © 2007 Published by Elsevier Inc.
doi:10.1016/j.laa.2007.02.029

Please cite this article in press as: F.T. Luk, D.M. Tracy, An improved LLL algorithm, Linear Algebra Appl. (2007), doi:10.1016/j.laa.2007.02.029
In Section 5, we extend the idea of a reduced basis to that of a reduced triangular matrix, and in Section 6, we present our new algorithm based on this extension. In Section 7, we show that the two methods will give the same results in exact arithmetic. We conclude the paper by presenting examples in Section 8 to illustrate how the numerical results produced by our new method can be significantly better than those produced by the original LLL method.

2. Integer least squares

Consider a linear least squares problem:

$$\min_s \| Bs - y \|_2,$$

where $B \in \mathbb{R}^{n \times n}$, $y \in \mathbb{R}^n$, $s \in \mathbb{Z}^n$, and $B$ is nonsingular. One important application is wireless communication: in a multiple-input-multiple-output (MIMO) model with a finite impulse response (FIR), the matrix $B$ could possess a block Toeplitz form. The problem (1) is NP hard; that is, the known solution algorithms all have exponential complexity. Indeed, most procedures are based on the Sphere Decoding Algorithm (SDA) of Pohst [5], which examines lattice points that lie inside a hypersphere. A two-step algorithm is given in Hassibi and Vikalo [3]:

1. Find the exact solution $B^{-1}y$ and round each element of the vector to the closest integer:
   $$\tilde{s} = \lceil B^{-1}y \rceil.$$
   The estimate $\tilde{s}$ is called a Babai point.

2. Use $\tilde{s}$ to determine the radius $\alpha$ of a sphere, and apply the SDA [5] to search over all points inside the sphere.

Note that the first step requires $O(n^3)$ flops and the second $O(\alpha n)$ flops. If the matrix $B$ has a special structure, such as a Toeplitz form, we could apply a fast QR decomposition technique and reduce the cost of step 1 to $O(n^2)$ flops. However, in light of the exponential cost in step 2, the saving is likely to be insignificant. To accelerate the convergence of SDA, Fincke and Pohst [1] suggested the use of the LLL algorithm.

To aid in the solution of (1), we use integer unimodular transformations:

**Definition 1.** A nonsingular matrix $M$ is unimodular if $\det(M) = \pm 1$.

**Lemma 1.** A nonsingular integer matrix $M$ is unimodular if and only if $M^{-1}$ is an integer matrix.

Given $B$, the idea in Lenstra et al. [4] is to construct a unimodular matrix $M \in \mathbb{Z}^{n \times n}$ so that the columns of $BM$ are almost orthogonal. The total work is $O(n^4)$. The integer least squares problem (1) then becomes

$$\min_s \|(BM)(M^{-1}s) - y\|_2. \quad (2)$$

Using LLL as a preconditioner to reduce the condition number of $BM$ in (2) is quite common in many applications; see, e.g., [3]. The preconditioning step is followed by a QR decomposition of $BM$ to solve the least squares problem. In Section 5, we will present a simpler and better approach that combines the two sequential steps into one single step by computing both $M$ and the QR decomposition of $BM$ at the same time.

Please cite this article in press as: F.T. Luk, D.M. Tracy, An improved LLL algorithm, Linear Algebra Appl. (2007), doi:10.1016/j.laa.2007.02.029
3. Reduced basis

Let \( B \in \mathbb{R}^{n \times n} \) be nonsingular. Consider its QR decomposition:

\[
Q^T B = DU,
\]

where \( Q \in \mathbb{R}^{n \times n} \) is orthogonal, \( D \equiv \text{diag}(d_i) \in \mathbb{R}^{n \times n} \) is diagonal with 
\[
d_i > 0 \quad \text{for} \quad i = 1, 2, \ldots, n
\]

and \( U \equiv (u_{ij}) \in \mathbb{R}^{n \times n} \) is upper triangular with ones on its diagonal:

\[
u_{ii} = 1 \quad \text{for} \quad i = 1, 2, \ldots, n.
\]

Note that instead of \( u_{ij} \), the parameter \( \mu_{ij} \) is used in [4]. Fortunately, the two parameters are related via

\[
u_{ij} = \mu_{ji} \quad \text{for all} \quad i \text{ and } j.
\]

A key concept in the LLL algorithm is that of a reduced basis.

Definition 2 [4]. The columns of \( B \) form a reduced basis if

\[
|u_{ij}| \leq 0.5 \quad \text{for} \quad 1 \leq i < j \leq n
\]

and

\[
d_i^2 \geq (\omega - u_{i-1,j}^2)d_{i-1}^2 \quad \text{for} \quad 2 \leq i \leq n,
\]

where \( 0.25 < \omega < 1 \) is a parameter that controls the rate of convergence.

Condition (4) states that the absolute value of any strictly upper triangular element of \( U \) is at most 0.5. Condition (5) states that the diagonal elements of \( D \) must be ordered in a certain manner. Let \( \omega = 0.75 \), a usual choice in [4]. Then (5) can be rewritten as

\[
d_i^2 \geq (0.75 - u_{i-1,j}^2)d_{i-1}^2 \geq (0.75 - 0.5^2)d_{i-1}^2 = 0.5d_{i-1}^2.
\]

Eq. (6) says that \( d_i^2 \) must be at least half as large as \( d_{i-1}^2 \).

Lemma 2. Since the value of the quantity inside the parentheses in (5) is always less than one, an upper triangular matrix \( B \in \mathbb{R}^{n \times n} \) with a constant diagonal satisfies condition (5).

Example 1. For \( 1 \leq i < j \leq n \), let \( u_{ij} \) denote any number so that \( |u_{ij}| \leq 0.5 \). The columns of this triangular matrix \( B_u \in \mathbb{R}^{n \times n} \) form a reduced basis:

\[
B_u = \begin{bmatrix}
1 & u_{12} & u_{13} & \cdots & \cdots & u_{1n} \\
1 & u_{23} & \cdots & \cdots & u_{2n} \\
& 1 & \cdots & \cdots & \cdots & u_{3n} \\
& & \ddots & \ddots & \ddots & \ddots \\
& & & \ddots & \ddots & \cdots & u_{n-1,n} \\
& & & & 1 & u_{nn} \\
& & & & & 1
\end{bmatrix}
\]
4. LLL reduction algorithm

In this section, we describe the actions of the LLL algorithm by showing how conditions (4) and (5) are enforced.

Condition (4) is easy to impose on $U \equiv (u_{ij})$, an upper triangular matrix with a unit diagonal. We begin by defining an elementary unimodular transformation. Let $i < j$, and let $e_i \in \mathbb{Z}^n$ and $e_j \in \mathbb{Z}^n$ denote unit coordinate vectors in the $i$th and $j$th directions, respectively. Define $M_{ij} \in \mathbb{Z}^{n \times n}$ by

$$M_{ij} \equiv I - \gamma e_i e_j^T,$$

where $\gamma$ is an integer.

**Lemma 3.** The matrix $M_{ij}$ defined in (8) is an integer unimodular transformation.

We use $M_{ij}$ to ensure that the $(i, j)$th element of $U$ is sufficiently small. Suppose that (4) is not satisfied for some $i$ and $j$; that is

$$|u_{ij}| > 0.5.$$

Calculate $\gamma$ as the integer closest to $u_{ij}$:

$$\gamma = \lfloor u_{ij} \rfloor.$$

Construct the unimodular matrix $M_{ij}$ with its $(i, j)$th element equal to $-\gamma$. Apply $M_{ij}$ to $B$ and $U$:

$$B \leftarrow BM_{ij} \quad \text{and} \quad U \leftarrow UM_{ij}.$$  (10)

It is straightforward to check that the $(i, j)$th element of the new $U$ satisfies (4).

**Procedure Decrease(i, j)** Given $B$ and $U$, calculate $M_{ij}$ and $\gamma$ using (8) and (9), respectively. Apply $M_{ij}$ to $B$ and to $U$:

$$B \leftarrow BM_{ij} \quad \text{and} \quad U \leftarrow UM_{ij}.$$  (12)

For condition (5), we need to define two numerical transformations.

**Notation 1.** The matrix $\Pi_i \in \mathbb{Z}^{n \times n}$ denotes a permutation in the $(i - 1, i)$ plane, where $2 \leq i \leq n$.

**Notation 2.** The matrix $X_i \in \mathbb{R}^{n \times n}$ denotes a transformation in the $(i - 1, i)$ plane, where $2 \leq i \leq n$. It has the form:

$$X_i \equiv \begin{bmatrix} I_{i-2} & \mu & 1 - \xi \mu \\ \mu & 1 - \xi & -\xi \mu \\ \xi & -\xi & I_{n-i} \end{bmatrix}.$$  (11)

Note that

$$\det(X_i) = -1$$  (12)
and that $X^{-1}_i$ is given by
\[
X^{-1}_i = \begin{bmatrix}
I_{i-2} & \xi & 1 - \xi \mu \\
1 & -\mu & \\
-\mu & I_{n-i}
\end{bmatrix}.
\] (13)

It is shown in [4] that the matrix $X^{-1}_i$ is made up of a product of two Gauss transformations [2]. Indeed, here is a quick illustration:
\[
\begin{bmatrix}
\xi & 1 - \xi \mu \\
1 & -\mu
\end{bmatrix} = \begin{bmatrix} 1 & \xi \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -\mu \end{bmatrix}.
\]
This matrix $X^{-1}_i$ is a workhorse in the LLL algorithm, and the following relation is key:
\[
\begin{bmatrix}
\xi & 1 - \xi \mu \\
1 & -\mu
\end{bmatrix} \begin{bmatrix} 1 & \mu \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & \xi \\ 0 & 1 \end{bmatrix}.
\] (14)

In words, Eq. (14) says that the matrix $X^{-1}_i$ restores the triangularity of a permuted triangular matrix. Note that both triangular matrices in (14) have ones on their diagonals.

Suppose that (5) is not satisfied for some $i$:
\[
d_i^2 < [\omega - u_{i-1,i}^2]d_{i-1}^2.
\]
We interchange columns $i$ and $i-1$ of $B$ and of $U$:
\[
B \leftarrow B \Pi_i \quad \text{and} \quad U \leftarrow U \Pi_i.
\] (15)
We then use the transformation $X^{-1}_i$ of (13) to restore $U$ to triangular form:
\[
U \leftarrow X^{-1}_i U.
\] (16)
The LLL paper [4] gives the formulas on updating the squares of the diagonal elements $d_{i-1}$ and $d_i$ of $D$. We skip the details and summarize the transformation in $D^2$ by
\[
D^2 \leftarrow D^2_{\text{new}}.
\] (17)
The paper [4] also gives the values of $\xi$ and $\mu$ in (11). As is obvious from (14), $\mu$ is given by
\[
\mu = u_{i-1,i}.
\] (18)
In addition, $\xi$ is given by
\[
\xi = \mu \cdot d_{i-1}^2/(d_i^2 + \mu^2 d_{i-1}^2).
\] (19)

PROCEDURE Swap(i) Given $D^2$, $B$ and $U$, update $D^2$, swap columns $i-1$ and $i$ of $B$ and of $U$, and use the transformation $X^{-1}_i$ to transform the permuted $U$ back to triangular form:
\[
D^2 \leftarrow D^2_{\text{new}}, \quad B \leftarrow B \Pi_i, \quad \text{and} \quad U \leftarrow X^{-1}_i U \Pi_i.
\] (20)
The matrix $X^{-1}_i$ is computed using Eqs. (11), (18), and (19).
We now present the LLL algorithm. A proof of convergence is given in [4].

ALGORITHM LLL Given $B$, transform its columns so that they will form a reduced basis.
compute QR decomposition of $B$ to get $D^2$ and $U$;
set $k \leftarrow 2$;
while $k \leq n$

if $|u_{k-1,k}| > 0.5$ then DECREASE $(k - 1, k)$;

if $d_k^2 < [\omega - u_{k-1,k}^2]d_{k-1}^2$ then

SWAP$(k)$;

$k \leftarrow \max(k - 1, 2)$;

else

for $i = k - 2$ down to 1

if $|u_{ik}| > 0.5$ then DECREASE$(i, k)$;

$k \leftarrow k + 1$.

It is well known (see [3] and references therein) that the LLL algorithm is an effective tool in reducing the condition number of a given matrix. However, LLL sometimes fails to decrease the condition number of an ill-conditioned matrix. We present one such example here.

**Example 2.** The LLL algorithm does not modify the matrix $B_u$ of (7) because its columns already form a reduced basis. Choose $u_{ij} = -0.5$ for all $i$ and $j$, and we get

$$
\hat{B} = \begin{bmatrix}
1 & -0.5 & -0.5 & \cdots & -0.5 \\
1 & -0.5 & \cdots & \cdots & -0.5 \\
1 & \ddots & \cdots & \cdots & \ddots \\
1 & \cdots & \ddots & \cdots & \ddots \\
1 & \cdots & \cdots & \cdots & -0.5 \\
1 & \cdots & \cdots & -0.5 & 1
\end{bmatrix}
$$

(21)

The matrix $\hat{B}$ is very ill-conditioned. Consider the matrix equation:

$$
\hat{B}x = e_n,
$$

where $n \geq 2$. The first element of the solution vector $x$ equals $(1.5)^{n-2}/2$. Thus, the smallest singular value of $\hat{B}$ decreases like $2(1.5)^{-n+2}$ as $n$ becomes large.

**5. A new idea**

We extend the idea of a reduced basis formed by the columns vectors to that of a reduced triangular matrix. Let $B \in \mathbb{R}^{n \times n}$ be nonsingular. Consider its QR decomposition:

$$
Q^T B = R, \quad (22)
$$

where $Q \in \mathbb{R}^{n \times n}$ is orthogonal and $R \equiv (r_{ij}) \in \mathbb{R}^{n \times n}$ is upper triangular with a positive diagonal:

$$
r_{ii} > 0 \quad \text{for } i = 1, 2, \ldots, p.
$$

The concept in [4] can be rewritten as follows.

**Definition 3.** The columns of $B$ form a reduced basis if

$$
r_{ii} \geq 2|u_{ij}| \quad \text{for } 1 \leq i < j \leq n \quad (23)
$$

and

$$
r_{ii}^2 \geq [\omega - (r_{i-1,i}/r_{ii})^2]r_{i-1,i-1}^2 \quad \text{for } 2 \leq i \leq n, \quad (24)
$$

where $0.25 < \omega < 1$ is a parameter that controls the rate of convergence.
Definition 4. A triangular matrix $R$ is reduced if its elements satisfy conditions (23) and (24).

Our extension will lead to a new algorithm to transform a given matrix $B$ to a reduced triangular matrix $R$.

Proposition 1. Given $B \in \mathbb{R}^{n \times n}$, our new algorithm generates an orthogonal matrix $Q \in \mathbb{R}^{n \times n}$ and a unimodular matrix $M \in \mathbb{Z}^{n \times n}$ to transform $B$ into a triangular matrix $R$:

$$Q^T BM = R,$$  (25)

so that $R$ is reduced. The columns of $BM$ form a reduced basis as defined in [4].

Our proposition will be proved by construction in the next section. We note that our new decomposition (25) is ideal for solving the integer least squares problem (2).

6. A new algorithm

In this section, we present our new algorithm and show how it enforces conditions (23) and (24). While condition (23) states that any diagonal element of $R$ is at least twice as large as any other element of $R$ along the same row, condition (24) states that the diagonal elements of $R$ must be ordered in a certain way. Let $\omega = 0.75$, a usual choice in [4]. Then (24) can be rewritten as

$$r_{ii}^2 \geqslant [0.75 - (r_{i-1,i}/r_{ii})^2]r_{i-1,i-1}^2 \geqslant [0.75 - 0.5^2]r_{i-1,i-1}^2 = 0.5r_{i-1,i-1}^2. $$  (26)

Eq. (26) says that $r_{ii}^2$ must be at least half as large as $r_{i-1,i-1}^2$.

We use $M_{ij}$ of (8) to ensure that the $(i, j)$th element of $R$ is sufficiently small. Suppose that (23) is not satisfied for some $i$ and $j$; that is

$$r_{ij} < 2|r_{ij}|.$$

Calculate $\gamma$ as the integer closest to $r_{ij}/r_{ii}$:

$$\gamma = \lceil r_{ij}/r_{ii} \rceil.$$  (27)

Construct the unimodular matrix $M_{ij}$ with its $(i, j)$th element equal to $-\gamma$. Apply $M_{ij}$ to $R$:

$$R \leftarrow RM_{ij}$$  (28)

and accumulate the transformations in $M$:

$$M \leftarrow MM_{ij}.$$  (29)

It is easy to check that the $(i, j)$th element of the new $R$ in (28) satisfies (23).

Procedure NewDecrease$(i, j)$ Given $R$ and $M$, calculate $M_{ij}$ and $\gamma$ using (8) and (27), respectively, and apply $M_{ij}$ to both $R$ and $M$:

$$R \leftarrow RM_{ij} \quad \text{and} \quad M \leftarrow MM_{ij}.$$  (30)

For condition (24) we need a basic numerical transformation [2].

Notation 3. The symmetric matrix $J_i \in \mathbb{R}^{n \times n}$ denotes a plane reflection in the $(i-1, i)$ plane, where $2 \leqslant i \leqslant n$. It has the form:

$$J_i \equiv \begin{bmatrix} I_{i-2} & c & s \\ c & s & -c \\ s & -c & I_{n-i} \end{bmatrix},$$  (31)

Please cite this article in press as: F.T. Luk, D.M. Tracy, An improved LLL algorithm, Linear Algebra Appl. (2007), doi:10.1016/j.laa.2007.02.029

RPI Geo* Final Report

Improved LLL Algorithm
where \( c^2 + s^2 = 1 \).

Note that
\[
\det(J_i) = -1, \quad (30)
\]
just like \( \det(X_i) = -1 \) in (12). In this paper, we use plane reflections instead of plane rotations because the \( X_i \)'s are closely related to plane reflections, as we will show in the next section.

Suppose that (24) is not satisfied for some \( i \):
\[
r_{i,i}^2 < \left[ \omega - (r_{i-1,i}/r_{i,i})^2 \right]r_{i-1,i-1}^2.
\]
We interchange columns \( i \) and \( i-1 \) of \( R \):
\[
R \leftarrow R \Pi_i \quad (31)
\]
and use a plane reflection \( J_i \) to restore \( R \) to triangular form:
\[
R \leftarrow J_i R. \quad (32)
\]
We accumulate the transformations in \( M \) and in \( Q \):
\[
M \leftarrow M \Pi_i \quad \text{and} \quad Q \leftarrow Q J_i. \quad (33)
\]

Now, we have all the tools to present our new algorithm as a matrix decomposition technique.

\textbf{Algorithm New} Given \( B \), compute \( M \), \( Q \), and \( R \), so that \( BM = QR \) and \( R \) is reduced.

\begin{itemize}
  \item compute \( B = QR \);
  \item set \( M \leftarrow I \) and \( k \leftarrow 2 \);
  \item while \( k \leq n \):
    \begin{itemize}
      \item if \( r_{k-1,k-1} < 2|r_{k-1,k}| \) then NEWDECREASE\((k - 1, k)\);
      \item if \( r_{k,k}^2 < \left[ \omega - (r_{k-1,k}/r_{k,k})^2 \right]r_{k-1,k-1}^2 \) then
        \begin{itemize}
          \item NEWSWAP\((k)\);
          \item \( k \leftarrow \max(k - 1, 2) \);
        \end{itemize}
      \item else
        \begin{itemize}
          \item for \( i = k - 2 \) down to \( 1 \)
            \begin{itemize}
              \item if \( r_{i,i} < 2|r_{i,k}| \) then NEWDECREASE\((i, k)\);
            \end{itemize}
          \item \( k \leftarrow k + 1 \).
        \end{itemize}
    \end{itemize}
\end{itemize}

7. \textbf{Comparing the two algorithms}

There are many similarities between Algorithms LLL and New. Both algorithms aim to reduce the given matrix \( B \) to a triangular form, and the overall structures are identical. The only difference lies in the transformations used: Algorithm New applies plane reflections \( J_i \) of (29) directly to \( R \), while Algorithm LLL applies special transformations \( X_i^{-1} \) of (13) to \( U \) and updates \( D^2 \) separately.

In this section, we will derive two \( n \times n \) diagonal matrices \( D_1 \) and \( D_2 \) such that
\[
J_i = D_1 X_i^{-1} D_2. \quad (34)
\]
Thus, we may view $X_i^{-1}$ as a scaled plane reflection. We will also show that in exact arithmetic, the two algorithms will produce identical numerical results.

Representing the effect of transformations (31) and (32) by

$$R_{\text{new}} = J_i R_i,$$

we write out the key $2 \times 2$ transformations as follows:

$$\begin{bmatrix} \hat{\alpha} & \hat{\gamma} \\ 0 & \hat{\beta} \end{bmatrix} = \begin{bmatrix} c & s \\ s & -c \end{bmatrix} \begin{bmatrix} \alpha & \gamma \\ 0 & \beta \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

(35)

from which we get

$$\begin{bmatrix} \hat{\alpha} & \hat{\gamma} \\ 0 & \hat{\beta} \end{bmatrix} = \begin{bmatrix} c & s \\ s & -c \end{bmatrix} \begin{bmatrix} \gamma & \alpha \\ \beta & 0 \end{bmatrix} = \begin{bmatrix} c\gamma + s\beta & c\alpha \\ s\gamma - c\beta & s\alpha \end{bmatrix}.$$  

(36)

Eq. (35) can be transformed into

$$\begin{bmatrix} 1 & \hat{\gamma}/\hat{\alpha} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\hat{\alpha} & 0 \\ 0 & 1/\hat{\beta} \end{bmatrix} \begin{bmatrix} c & s \\ s & -c \end{bmatrix} \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$ (37)

Define a new transformation $Y$ by

$$Y \equiv \begin{bmatrix} 1/\hat{\alpha} & 0 \\ 0 & 1/\hat{\beta} \end{bmatrix} \begin{bmatrix} c & s \\ s & -c \end{bmatrix} \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}.$$ (38)

Then

$$\begin{bmatrix} 1 & \hat{\gamma}/\hat{\alpha} \\ 0 & 1 \end{bmatrix} = Y \begin{bmatrix} 1 & \gamma/\alpha \\ 0 & 1 \end{bmatrix}$$

(39)

and

$$Y = \begin{bmatrix} c\alpha/\hat{\alpha} & s\beta/\hat{\alpha} \\ s\alpha/\hat{\beta} & -c\beta/\hat{\beta} \end{bmatrix} = \begin{bmatrix} \gamma/\alpha & (\alpha - c\gamma)/\hat{\alpha} \\ \beta/\hat{\beta} & -s\gamma/(s\alpha) \end{bmatrix} = \begin{bmatrix} \hat{\gamma}/\hat{\alpha} & 1 - \gamma\gamma/(\hat{\alpha}\alpha) \\ \beta/\hat{\beta} & -s\gamma/(s\alpha) \end{bmatrix}.$$ (40)

by using the equalities in (36). If we choose

$$\xi = \hat{\gamma}/\alpha \quad \text{and} \quad \mu = \gamma/\alpha,$$

(41)

then we get

$$Y = \begin{bmatrix} \xi & 1 - \xi\mu \\ 1 & -\mu \end{bmatrix}.$$ (42)

and

$$\begin{bmatrix} 1 & \xi \\ 0 & 1 \end{bmatrix} = Y \begin{bmatrix} 1 & \mu \\ 0 & 1 \end{bmatrix}.$$ (43)

Note that (43) is exactly Eq. (14) for the LLL method. Also, we can easily prove that the $\mu$ and $\xi$ as defined in (41) have the same values as the $\mu$ and $\xi$ as defined in (18) and (19). Thus, the transformation $Y$ is exactly the $2 \times 2$ part of the workhorse $X_i^{-1}$ of the LLL algorithm.

From (37), we get

$$\begin{bmatrix} c & s \\ s & -c \end{bmatrix} = \begin{bmatrix} \hat{\alpha} & 0 \\ 0 & \hat{\beta} \end{bmatrix} Y \begin{bmatrix} 1/\alpha & 0 \\ 0 & 1/\beta \end{bmatrix}.$$ (44)
Let
\[
D = \begin{bmatrix}
E_1 & \alpha & 0 \\
0 & 0 & \beta \\
\end{bmatrix},
\] (40)
where \(E_1 \in \mathbb{R}^{(i-2)\times(i-2)}\) and \(E_2 \in \mathbb{R}^{(n-i)\times(n-i)}\) are positive diagonal matrices. Define
\[
D_1 = \begin{bmatrix}
E_1 & \hat{\alpha} & 0 \\
0 & 0 & \hat{\beta} \\
\end{bmatrix}
\] and
\[
D_2 = \begin{bmatrix}
E_1^{-1} & 1/\alpha & 0 \\
0 & 1/\beta & E_2^{-1} \\
\end{bmatrix}.
\] (41)

Then
\[
J_i = D_1 X_i^{-1} D_2.
\] (42)

Consider
\[
J_i R = D_1 X_i^{-1} D_2 R.
\]

We see that \(D_2\) reduces \(R\) to a unit-diagonal triangular matrix (namely \(U\)), and that \(D_1\) gives the new diagonal of \(D_2 R\) after the transformation by \(X_i^{-1}\). Therefore, we conclude that Algorithms LLL and New produce the same numerical results in exact arithmetic. It also follows that the convergence result for Algorithm LLL in [4] is applicable to Algorithm New.

The LLL algorithm [4] is numerically efficient in that it avoids the computation of square roots, which is one reason why it updates \(D^2\) instead of \(D\). Thus, we may view the transformations in the LLL method as square-root-free plane reflections. The potential cost for this efficiency is a possible loss in numerical accuracy when the given matrix is ill-conditioned, as we shall show in the next section.

8. Numerical experiments

In this section, we present numerical examples to compare our new method against the original LLL algorithm. The initial matrix \(B \in \mathbb{R}^{n\times n}\) is upper triangular with each nonzero element as a random number in \((-1, 1)\). We use the symbol \(\kappa\) to represent the condition number of a matrix.

Thus,
\[
\kappa(B) \equiv \text{cond}(B).
\]

For well-conditioned test matrices, the two different schemes produce essentially identical results. Hence we show mostly ill-conditioned examples in Table 1. However, to avoid matrices that are numerically singular, we place an upper limit on the condition number of \(B\):
\[
\kappa(B) \leq 10^{15};
\]

that is, we would keep on generating test matrices until we get one matrix that has a sufficiently small condition number. To see which method is better, we compare the two resultant triangular matrices \(DU\) and \(R\), and the condition numbers of the two resultant \(BM\)'s. In exact arithmetic, \(DU\) should equal \(R\). We therefore calculate the Frobenius norm of the difference:
\[
\|DU - R\|_F,
\]
and normalize the result by dividing by the quantity \(\alpha(n)\), given by
\[
\alpha(n) = \sqrt{n(n+1)/2}.
\]
Our proposal that our method works better than the original LLL method is also supported by the values of $\kappa(BM)$. In Table 1, we observe that $\kappa_{\text{LLL}}(BM) > 10^3$ for $n \geq 50$ and $\kappa_{\text{New}}(BM) < 10^3$ for $n \leq 100$.

Indeed, when $n = 100$, we get

\[ d_{\text{max}} = 32.78 \quad \text{and} \quad r_{\text{max}} = 0.84 \]

and

\[ \kappa_{\text{LLL}}(BM) = 1.53 \times 10^9 \quad \text{and} \quad \kappa_{\text{New}}(BM) = 6.47 \times 10^2. \]

Thus, our experimental results confirm our theory that our method should produce more accurate results than the LLL method because orthogonal transformations are more stable than Gauss transformations.

Please cite this article in press as: F.T. Luk, D.M. Tracy, An improved LLL algorithm, Linear Algebra Appl. (2007), doi:10.1016/j.laa.2007.02.029
Acknowledgements

The authors acknowledge helpful discussions with Professors Richard Brent and Joohwan Chun. They are grateful to Professor Brent for many valuable suggestions to improve the paper.

References

Surface Compression using Over-determined Laplacian Approximation

Zhongyi Xie\textsuperscript{1}, W. Randolph Franklin\textsuperscript{2}, Barbara Cutler\textsuperscript{1}, Marcus A. Andrade\textsuperscript{2,3}, Metin Inanc\textsuperscript{1} and Daniel M. Tracy\textsuperscript{1}

\textsuperscript{1}Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York, 12180–3590, USA;
\textsuperscript{2} Department of Electrical, Computer, and System Engineering, Rensselaer Polytechnic Institute, Troy, New York, 12180–3590, USA
\textsuperscript{3} Universidade Federal de Viçosa, Viçosa, Brazil

ABSTRACT

We describe a surface compression technique to lossily compress elevation datasets. Our approach first approximates the uncompressed terrain using an over-determined system of linear equations based on the Laplacian partial differential equation. Then the approximation is refined with respect to the uncompressed terrain using an error metric. These two steps work alternately until we find an approximation that is good enough. We then further compress the result to achieve a better overall compression ratio. We present experiments and measurements using different metrics and our method gives convincing results.

1. INTRODUCTION

Nowadays, the size of digital terrain data has grown to an extent that makes it essential to use some special representation or compression technique to manipulate the data. For example, LIDAR (Light Detection and Range), which is a laser based range detector coupled with a GPS sensor, allows for 20,000 to 50,000 readings per second. Each reading is stored as an \( \{x, y, z\} \) triplet where each coordinate is represented as an IEEE double precision floating point number amounting for 24 bytes per point. LIDAR was the technology used for the state of North Carolina after Hurricane Floyd (1999) to map the whole state in the NC Floodplain Mapping Project. Just the Neuse river basin (11\% of the whole NC area) is made of approximately 500 million points and takes 11.2 GB to store.\textsuperscript{1,2}

However, the development of processing and handling of digital terrain data has not advanced in pace with the data inflation. Elevation datasets are still stored as an elevation matrix where common compression algorithms do not perform very well, and there are not many algorithms specifically designed to compress terrain data. For example, USGS DEM data are usually compressed with gzip, which was originally designed as a plain text compressor.\textsuperscript{1}

In this paper, we use Over-determined Laplacian Partial Differential Equations (ODETLAP) to approximate and lossily compress terrains. By terrain we mean a single-valued elevation matrix, which excludes caves or overhangs. We construct an over-determined system using triangulation, visibility tests, level set components and random selection; then use an over-determined PDE to solve for a smooth approximation. The initial approximation might be very rough, i.e., it may contain considerable elevation or slope errors. After that, we refine the approximation with respect to the original terrain by adding into the important points set those points with biggest elevation error or slope error, and then use ODETLAP again on the augmented representation.
to find a better approximation. These two steps are alternately applied until a predefined maximum error is reached. We also study the size versus accuracy tradeoff and plot the error curve.

ODETLAP can process not only continuous contour lines but isolated points as well. The surface produced tends to be smoother while preserving high accuracy to the known points. Local maxima are also well preserved. Alternative methods generally sub-sample contours due to limited processing capacity, or ignore isolated points.

This paper is organized as follows. In section 2 we briefly walk through current terrain representation and compression techniques. In section 3.1, we describe in detail the definition of ODETLAP, followed by section 3.2 which covers the outline of the algorithm that compresses the digital terrain data. In section 4, we describe several different ways to select points that will be used by ODETLAP to reconstruct the terrain. Section 5 presents details on encoding the points to maximize the compression ratio. Finally in section 6 and section 7 we talk about results and ideas for future work.

2. RELATED WORK

Simplification and compression of three dimensional terrain/surface data share the same goal but take different routes: simplification reduces the complexity of the terrain by reducing the number of vertices and faces in the mesh while compression works to compactly represent the connectivity data of the terrain. We will briefly discuss some classical work in both areas.

2.1 Progressive Meshes

The Progressive meshes (PM) is a surface representation introduced by Hoppe.\textsuperscript{3} The simplification is based on a pair of reversible operations: edge collapse and vertex split, the first of which works for simplification and the latter for reconstruction/refinement. A PM maintains a sequence of refinement/simplification records so that a mesh of any precision can be obtained by incremental refinement.

2.2 Triangulated Irregular Network

The Triangulated Irregular Network (TIN), a piecewise linear triangular spline, first implemented in cartography in 1973\textsuperscript{4} is an approximated, lossy representation that has the major advantage that it is not tied either to a particular coordinate system, or to a developable surface. We use an implementation which is greedy: find the point with biggest error and use it to refine the triangulation. For more information, please see section 4.1.2.

2.3 Mesh Compression

Taubin and Rossignac propose a mesh compression scheme which records the connectivity information into a spanning tree.\textsuperscript{5} Their method is capable of compressing connectivity information to 2 bits per triangle. Normals, colors, and texture coordinates, are compressed in a similar manner. One of the disadvantages of this method is its large memory requirement due to the requirement of random access to all vertices in decompression.

2.4 Hierarchical Triangulation

Hierarchical Triangulation (HT) is a hierarchical triangle-based model for representing surfaces over sampled data proposed by De Floriani.\textsuperscript{6} Similar to TIN, HT is based on subdividing the surface into nested triangulations which is then organized into a tree where each node stands for a triangulation except the root. In order to describe the surface of each triangles, HT associates values with vertices of the triangles. HT is capable of extracting the representation of terrain at variable resolutions over the domain.

2.5 Visibility Preserving Terrain Simplification

Ben-Moshe proposes a terrain simplification technique based on preserving inter-point visibility relationships.\textsuperscript{7} This technique aims at preserving visibility information, which, informally speaking, means points in the original terrain that are visible to each other (and respectively, points not visible to each other) are still visible to each other (respectively, not visible to each other) after the simplification. This technique is designed to meet the need of finding good locations on the terrain to place “observers” (antennas, guards, etc). It works by first computing the ridge network (a collection of chains of edges of the terrain). This ridge network induces a subdivision of the terrain into patches and each patch is independently simplified using one of the standard terrain simplification methods.
3. OVER-DETERMINED LAPLACIAN APPROXIMATION

3.1 Definition

As implied by the name, the Over-determined Laplacian Approximation (ODETLAP) comes from Laplace’s equation, whose solution at any point \((x, y, z)\) is equal to the average of the solution values on the surface of any sphere with center \((x, y, z)\), assuming the equation holds throughout the sphere.\(^8\) We have the equation

\[
4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1}
\]  

(1)

for every unknown non-border point, which is equivalent to saying the surface satisfies Laplacian PDE,

\[
\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = 0
\]  

(2)

In terrain modeling this equation has the following limitations:

- The solution of Laplace’s equation never has a relative maximum or minimum in the interior of the solution domain, this is called the “maximum principle”\(^8\), so local maxima are never generated.
- The generated surface may droop if a set of nested contours is interpolated\(^9\)

To avoid these limitations, an over-determined version of the Laplacian equation is defined as follows: apply the equation (1) to every non-border point, both known and unknown, and a new equation is added for a set \(S\) of known points:

\[
z_{ij} = h_{ij}
\]  

(3)

where \(h_{ij}\) stands for the known elevations of points in \(S\) and \(z_{ij}\) is the “computed” elevation for every point, like in equation (1).

Note that the system of linear equations is over-determined, i.e., the number of equations exceeds the number of unknown variables. Since the system is very likely to be inconsistent, instead of solving it for an exact solution (which is now impossible), an approximated solution is obtained by trying to keep the error as small as possible. Equation (1) is satisfied for each point, making it the average of its neighbors, which makes the generated surface smooth there. However, since we have known points where equation (3) is valid, they are not necessarily equal to the average of their neighbors, which probably make the surface not smooth there. This is especially true when we have adjacent known points, like points that defines contour lines. Therefore, for points with multiple equations we can choose the relative importance of accuracy versus smoothness by adding a smoothness parameter when solving the over-determined system.\(^{10}\) In our implementation, equation (1) is weighted by \(R\) relative to equation (3) which defines the known locations. So a very small \(R\) will approximate a determined solution and the surface will be more accurate while a very large \(R\) will produce a surface with no divergence, effectively ignoring the known points. figure 2 shows how different values of \(R\) will affect the generated surface.\(^4\) Subfigure (a) shows the four nested square contour lines that we try to approximate. Subfigure (b) gives the Lagrangian interpolation and we can see the undesired lines that surface normal is not continuous. Subfigure (c) and (d) are generated by ODETLAP and with \(R\) equal to 1 and 10. So in (c) where we made the accuracy as important as smoothness, the surface is quite accurate compared to in subfigure (d). However, the visible contours means it’s not as smooth as subfigure (d).

This over-determined system allows for processing of isolated, scattered elevation points as well as continuous contour lines and produces a smooth surface while the error is minimized. The generated surface has local maxima inside the innermost contour and shows little or no evidence of the contours. Instead of interpolation, approximation is a more suitable term for this method because the reconstructed surface is not guaranteed to go through the input data points.

ODETLAP can be used as a lossy compression technique since the original terrain can be approximated with some error using the set of points \(S\) for equations (1) and (3).
3.2 Algorithm Outline

The ODETLAP algorithm’s outline is shown in figure 1 and the pseudo code is given below. Starting with the original terrain elevation matrix there are two point selection phases: firstly, the initial point set \( S \) is built by any of the methods described in section 4.1 and a first approximation is computed using the equations (1) and (3). Given the reconstructed surface, a stopping condition based on an error measure is tested. In practice, we have used the root-mean-square (RMS) error as the stopping condition. If this condition is not satisfied, the second step is executed. In this step, \( k \geq 1 \) points from the original terrain are selected by method described in 4.2 and they are inserted in the existing point set \( S \); this extended set is used by ODETLAP to compute a more refined approximation. As the algorithm proceeds, the total size of point set \( S \) increases and the total error converges.

\[
\begin{align*}
\text{input} & \quad \text{OriginalTerrain}: T \\
\text{output} & \quad \text{PointSet}: S \\
0 & \quad S = \text{InitSelection}(T) \\
1 & \quad \text{Reconstructed} = \text{ODETLAP}(S) \\
2 & \quad \text{while } \text{RMS(Reconstructed)} > \text{Max_RMS} \\
3 & \quad S = S \cup \text{Refine}(T, \text{Reconstructed}) \\
4 & \quad \text{Reconstructed} = \text{ODETLAP}(S) \\
5 & \quad \text{return } S
\end{align*}
\]

4. POINT SELECTION STRATEGIES

As we have seen in section 3.2, there are two stages where points are selected: the initial point selection stage and the refined point selection. We discuss each of them below.

4.1 Initial Points Selection

4.1.1 Random Selection

This strategy is the most intuitive and easiest to implement. The basic idea is select points randomly. Using a good random number generator, this strategy ensures that most parts of the terrain contribute to the final reconstruction. This strategy is fast and robust.

4.1.2 Triangulated Irregular Network

The next strategy is based on the Franklin’s algorithm\(^4\) which builds a triangulated irregular network (TIN) using a greedy insertion method to approximate a surface. Starting with a matrix of elevations, it first splits the points bounding square (or rectangle) into two triangles along the diagonal and associates each triangle with its points. In the next step, it searches within each triangle for the furthest point from the triangle’s plane and this point is used to split the triangle into three new triangles (or two new triangles if the furthest point happens to be on an edge of the triangle). It uses a breadth first search to avoid starvation: a triangle is never split if there exists an undivided triangle that was created before it. An issue with this approach is that in some steps the insertion of the furthest point may temporarily increase the error, but usually, after some additional insertions, the error will be reduced even more. So, the overall tendency is for the error to decrease when new points are added.
4.1.3 Visibility Index

The visibility index (VIX)\textsuperscript{11} of a point $p$, the “observer”, on the terrain is defined as the number of terrain points that $p$ can see. An approximated version of this index is defined considering only a small region around $p$; generally, this region is given by a circle centered at $p$ with a radius (of interest) $r$.

There are several ways to compute the VIX values and we use the one proposed by Ray and Franklin\textsuperscript{11} for each terrain point $p$, randomly select $k$ sample points within the radius of interest and run a line of sight connecting $p$ to each random point to decide if the point is visible or not. The VIX of $p$ is given by the ratio

\begin{equation}
    \text{VIX}(p) = \frac{\text{Number of visible points}}{k}
\end{equation}
between the number of visible random points and sample size. As a result, the VIX values are only approximation
to the exact values and they are highly dependent on how many random points are chosen. In our tests, we used
\( r = 25 \) and \( k = 10 \).

In this point selection strategy we assume that points with small VIX values are more important to define
the terrain skeleton than points with big VIX. So the initial set is built containing points with small VIX values.
However, our selection of points need to reflect the overall VIX value distribution. This is done using a probability
distribution that establishes how likely a point with a determined VIX value will be selected. This probability
distribution is defined using the VIX value (small values mean bigger probabilities) weighted by the VIX values
distribution. Thus, points whose VIX value occurs more frequently have their probability multiplied by a higher
factor. Figure 4 shows the selected points from the terrain.

![Figure 4. Visibility Index: Points are selected according to their visibility indices](image)

4.1.4 Level set components

Using an adaptation of level set ideas, we segment the terrain based on points’ elevation. That is, suppose that
the elevation values range from \( h_{\text{min}} \) to \( h_{\text{max}} \) and given an integer \( k \), the interval \([h_{\text{min}}, \ldots, h_{\text{max}}]\) is divided into
“elevation slices” of equal size \( k \) (the last slice can be smaller). Then, each terrain point \( p = (i, j, h) \) is associated
to the corresponding elevation slice that contains the height \( h \); more precisely, to the slice \([h_i, \ldots, h_{i+1}]\) such that
\( h_i \leq h < h_{i+1} \). Next, each elevation slice is partitioned into connected components which are computed using
8-connectivity, (i.e., the horizontal, vertical and diagonal neighbors are checked)\(^*\)

As in the previous strategy, this point selection criterion uses a probability distribution defined considering
the elevation slices area (i.e., the total number of points in all the connected components in the elevation slice)
weighted by an “elevation slice importance” assigned assuming that the most important slices are those in the
extremities (lowest and highest) height - the importance decreases uniformly toward the slice with medium
height.

4.2 Refined point selection - Greedy algorithm

After the initial point set is obtained, ODETLAP is used to reconstruct the elevation matrix. This matrix has
high error with respect to the original terrain, mostly due to the limited size of the initial point set. As shown in
figure 1, refined points selection is applied and a set of additional points is chosen and added to the existing points
set \( S \) to form the augmented points set. The way we choose new points is greedy (similar to 4.1.2): we find a set
of points with greatest absolute vertical error. The size of the set in our experiments is intentionally kept small
(10% or smaller) so that for a given total number of points, more iterations could be used to reduce the error as
much as possible. This is actually a trade-off between accuracy and computation time. The augmented set \( S' \)

\(^*\)Of course, the elevation slice point association and the connected component computation can be done simultaneously
just adapting the connected component computation to check if a neighbor is in the same elevation slice.
is then given to ODETLAP to reconstruct a more refined approximation. The newly obtained approximation is again examined with respect to the original terrain against our stopping condition, which is either:

1. Relative RMS: Compute the root mean square (RMS) error of the approximation and check if its ratio against the RMS error of the first approximation is smaller than a predefined threshold.

or

2. Absolute RMS: Define a value for the maximum acceptable RMS error.

### 4.3 Forbidden Zone

![Forbidden Zone](image)

**Figure 5. Forbidden Zone**: Points are often clustered as in the left figure, hence reconstructed surface is not very accurate; We apply forbidden zone in refined points selection so that points are no longer clustered as in the right figure and reconstructed surface is more accurate as we can see in table 1.

Using the refined point selection described in section 4.2, one can encounter a problem: the refined points are sometimes clustered (left figure in figure 5). This is because real terrains are mostly continuous so if one point is far away, adjacent points are also likely to be erroneous, and will be selected as well. Because of this, refined points selected by any of our strategies may be redundant in some regions, which is a waste of storage.

![Forbidden Zone](image)

**Figure 6. Forbidden Zone**

We perform a check process when adding new refined points: the local neighbor of the new point is checked to see if there is any existing refined points which were added in the same iteration. If yes, this new refined point is discarded and point with the next biggest error is tested until we find desired number of refined points. So as shown in figure 6, all potential refined points that are close to an existing refined point (green points) are useless (marked red), and only points that are beyond some distance from green points are selected (marked yellow). The effect of forbidden zone can be seen in the right figure in figure 5: no dense clusters of points are present and all points are distributed more evenly within the whole terrain.
### Table 1. Impact of forbidden zone: Size of forbidden zone = 5.

<table>
<thead>
<tr>
<th>Data</th>
<th>Avg Error(w/F.Z.)</th>
<th>Avg Error(w/o F.Z.)</th>
<th>Max Error(w/F.Z.)</th>
<th>Max Error(w/o F.Z.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1</td>
<td>3.16</td>
<td>7.35</td>
<td>19.9</td>
<td>34.3</td>
</tr>
<tr>
<td>Hill2</td>
<td>8.34</td>
<td>14.8</td>
<td>48.9</td>
<td>71.6</td>
</tr>
<tr>
<td>Hill3</td>
<td>1.54</td>
<td>3.00</td>
<td>10.3</td>
<td>13.5</td>
</tr>
</tbody>
</table>

In table 1, we show how the forbidden zone affects the average error and maximum error in our experiment. We implemented our algorithm with and without the forbidden zone and tested it with three data sets. In this test, we begin with 300 points and in each iteration 50 refined points are added. The results after 14 iterations justify the use of the forbidden zone in our algorithm.

### 5. FURTHER COMPRESSION - COMPRESSING REFINED POINTS

In order to achieve better compression, we apply data compression to the final point set \(S\), an \(n \times 3\) matrix where each row represents a point, and the three columns represent \(x, y\) and \(z\). Our approach is to separate the first two columns \(\{x, y\}\) from the last column \(\{z\}\), because \(x\) and \(y\) are integer coordinates (matrix rows and column indices) while \(z\) is the elevation value. Unlike \(x\) and \(y\) which are uniformly distributed over \([1, N]\), where \(N^2\) is the size of the dataset, the elevation values tend to have a smaller deviation. This makes possible a more compact compression if a prediction scheme is used. To preserve the \(x, y, z\) correlation, we either sort the \(n \times 3\) matrix on \(\{x\ and \ y\}\) or \(\{z\}\) prior to splitting it into a \(n \times 2\) matrix and a \(n \times 1\) matrix. As expected, different sorting orders give different compression sizes.

In order to achieve higher compression, we use linear prediction and delta coding, which predicts the next value using the previous one. To illustrate, suppose we have a vector \((954, 1021, 1001, 897, 958, 1130)\). The predicted vector would be \((0, 954, 1021, 1001, 897, 958)\), but we only need to store \(\Delta\), the difference between the predicted and actual values: \((954, -67, -20, -104, 61, 172)\). Compressing the difference using a data compressor like bzip2 further reduces the size of the compressed data.

To illustrate the effect of sorting and linear prediction, we have tested the following five schemes, and their compressed sizes are given in figure 7:

1. Sort rows on \(\{x\ and \ y\}\), do not use linear prediction.
2. Sort rows on \(\{x\ and \ y\}\), use linear prediction only in \(\{z\}\).
3. Sort rows on \(\{x\ and \ y\}\), use linear prediction in both \(\{x, y\}\) and \(\{z\}\).
4. Sort rows on \(\{z\}\), use linear prediction only in \(\{z\}\).
5. Sort rows on \(\{z\}\), use linear prediction in both \(\{x, y\}\) and \(\{z\}\).

From the figure, we can see that techniques 3, 4 and 5 give smaller compressed sizes. Comparing bzip2 on the original data set to the best technique, we see a 33\% improvement.

### 6. RESULT AND ANALYSIS

We have tested our algorithm on six different data sets. The results are presented in Table 2 showing that our new compression scheme is doing well in lossily compressing the terrain. We can successfully compress them from binary size of 320KB to less than 3KB, while keeping the mean absolute error under 2\% in all cases.
Table 2. ODETLAP Compression results: Each of the ODETLAP tests consist of 100 initial points selected with the TIN method, and then 10 points are added using the greedy selection method on each iteration for 90 iterations, for a total of 1000 points. Forbidden zone is used and we chose $R = 0.01$ in all cases.

### 6.1 Impact of points selection strategies

The approximation error obtained in the method’s first step varies a lot depending on how the initial points are selected. We have tested all strategies mentioned in section 4.1, to determine which strategy works best, i.e., discovers the most important points that define the terrain profile such that the approximation error would be lowest.

We have tested the four initial points selection methods described in section 4.1. In the experiments, we used different number of initial points for all the methods and recorded the RMS, average absolute and maximum absolute errors of the reconstructed surface. As we can see in the figure 8, the four methods’s performance is quite similar in terms of RMS errors and average errors, while TIN is always slightly better when number of points is greater than 300. In table 3, we also have the RMS errors of the four methods, and this corresponds to the first subgraph in figure 8. We notice that TIN is generating the most accurate surface with approximately 20% better than any of other methods. Surprisingly, random is also a competitive method: it’s error is not too much larger than any other method. Considering its efficiency, robustness and simplicity, it is also a good choice besides TIN.

### 6.2 Error vs. Size

As we have seen in previous discussions, our ODETLAP based algorithm refines the approximated surface as more points are added. In order to figure out how the error decreases as number of refined points increases, we experiment by starting with 10 initial points selected by TIN method and adding 1 refined point each time so as to get a complete error/size curve; Please refer to figure 9 for more information.
Figure 8. Comparison of initial points selection methods using different number of initial points: RND - random selection strategy, TIN - Triangulated irregular network, VIX - visibility index, LSC - the level set components.

<table>
<thead>
<tr>
<th># of Points</th>
<th>TIN</th>
<th>VIX</th>
<th>RND</th>
<th>LSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>419.1</td>
<td>171.7</td>
<td>114.0</td>
<td>141.5</td>
</tr>
<tr>
<td>64</td>
<td>369.5</td>
<td>79.15</td>
<td>92.72</td>
<td>108.3</td>
</tr>
<tr>
<td>128</td>
<td>89.08</td>
<td>69.81</td>
<td>82.17</td>
<td>58.98</td>
</tr>
<tr>
<td>160</td>
<td>78.18</td>
<td>61.72</td>
<td>67.42</td>
<td>57.63</td>
</tr>
<tr>
<td>320</td>
<td>47.37</td>
<td>51.13</td>
<td>46.54</td>
<td>44.00</td>
</tr>
<tr>
<td>480</td>
<td>39.50</td>
<td>38.92</td>
<td>39.00</td>
<td>37.50</td>
</tr>
<tr>
<td>640</td>
<td>31.02</td>
<td>36.72</td>
<td>34.38</td>
<td>33.07</td>
</tr>
<tr>
<td>800</td>
<td>27.06</td>
<td>36.21</td>
<td>30.75</td>
<td>30.41</td>
</tr>
<tr>
<td>960</td>
<td>22.70</td>
<td>28.43</td>
<td>28.03</td>
<td>27.88</td>
</tr>
<tr>
<td>1120</td>
<td>20.54</td>
<td>27.43</td>
<td>25.40</td>
<td>24.63</td>
</tr>
<tr>
<td>1280</td>
<td>18.86</td>
<td>24.44</td>
<td>23.65</td>
<td>23.33</td>
</tr>
<tr>
<td>1440</td>
<td>18.26</td>
<td>23.18</td>
<td>22.34</td>
<td>22.02</td>
</tr>
<tr>
<td>1600</td>
<td>17.21</td>
<td>21.88</td>
<td>21.11</td>
<td>21.67</td>
</tr>
<tr>
<td>2400</td>
<td>15.25</td>
<td>17.43</td>
<td>17.03</td>
<td>16.52</td>
</tr>
</tbody>
</table>

Table 3. Comparison initial points selection methods:

7. CONCLUSION AND FUTURE WORK

We create a new terrain compression technique based on an Overdetermined Laplacian equation that computes good compression results. We use several points selection methods as to find most important points. In order to get better results, we also use forbidden zone in selecting refined points as well as delta encoding in further compression and both of them work well. Moreover, our compression technique does not only work well in conventional error metrics as we have seen in table 2, but some other application specific error metrics as well.
Figure 9. Both average absolute error and maximum absolute error decrease as more points are added: y axis stands for errors and x axis stands for total number of points.

For example, as described in, more complex metrics such as visibility and path planning are employed to evaluate the reconstructed approximation from ODETLAP and the results are also very good.

The next step of research consists of a few extensions in two directions: one is higher accuracy. We will investigate other PDEs to see if they can reconstruct the terrain more accurately than the Laplacian PDE. Another direction is higher compression. Currently we use lossless compression in the final compression step. We will test the use of lossy schemes, which can reach higher compression ratios at the cost of loss in accuracy. Since slope is also a very important feature of terrain, we will also consider ways to minimize slope error in our representation. For example, we will investigate selecting refined points based on largest slope error instead of elevation error or a combination of the two.

8. ACKNOWLEDGEMENTS

This research was supported by NSF grants CCR-0306502 and DMS-0327634, by DARPA/DSO/GeoStar, and by CNPq - the Brazilian Council of Technological and Scientific Development. We thank Prof. Franklin T. Luk, Jonathan Muckell and Jared Stookey for valuable discussions on terrain representation.

REFERENCES


Multiple Observer Siting and Path Planning on a Compressed Terrain

Daniel M. Tracy\textsuperscript{a}, W. Randolph Franklin\textsuperscript{b}, Barbara Cutler\textsuperscript{a}, Marcus Andrade\textsuperscript{b,c}, Franklin T. Luk\textsuperscript{a}, Metin Inanc\textsuperscript{a}, and Zhongyi Xie\textsuperscript{a}

\textsuperscript{a}Department of Computer Science, Rensselaer Polytechnic Institute; \textsuperscript{b}Department of Electrical, Computer, & Systems Engineering, Rensselaer Polytechnic Institute \textsuperscript{c}Universidade Federal de Viçosa, Viçosa, Brazil

ABSTRACT

We examine a smugglers and border guards scenario. We place observers on a terrain so as to optimize their visible coverage area. Then we compute a path that a smuggler would take so as to avoid detection, while also minimizing the path length. We also examine how our results are affected by using a lossy representation of the terrain instead.

We propose three new application-specific error metrics for evaluating terrain compression. Our target terrain applications are the optimal placement of observers on a landscape and the navigation through the terrain by smugglers. Instead of using standard metrics such as average or maximum elevation error, we seek to optimize our compression on the specific real-world application of smugglers and border guards.

1. INTRODUCTION

The usual way to evaluate a compression scheme is to calculate the per-pixel average and the maximum errors between the original and reconstructed geometry. This metric may not be appropriate for the common tasks performed on terrain data. It may be beneficial to consider more domain-specific applications. For example, visibility is can be useful for surveying, cell phone tower placement, military surveillance, etc.

We consider the problem of multiple observer siting on a compressed terrain. An observer is placed at a point on the terrain and can see other points on the terrain only if no other part of the landscape obstructs a direct line of sight to the points, and the point is within the radius of visibility. Typically, the observer is placed at a specified height above the terrain, and lines of sight are drawn out to targets that are also a specified height above the terrain. In a highly mountainous terrain, points within narrow valleys typically have low visibility. The problem is to site a group of observers so as to maximize the amount of visible terrain; an application is the placement of watchtowers to observe a territory. In many real world scenarios, this problem must be solved using only the compressed terrain because the original terrain requires too much storage. For our problem, we want to evaluate the quality of the reconstructed geometry provided by the compression scheme.

In this paper, we propose a new test protocol and new error metrics. Our new protocol is to compute a minimum-length path from the northwest corner of the terrain to the southeast corner, while avoiding detection, and our new error metrics are to examine path lengths and visibility errors.

2. TERRAIN COMPRESSION TECHNIQUES

We consider a novel compression technique that we have developed, ODETLAP. We also examine JPEG, which was originally developed for image compression, but it has been repurposed and applied to other compression tasks. However, artifacts that are acceptable in the image domain could be problematic in terrain compression.
2.1 ODETLAP

Franklin, Xie, and Inanc developed an alternate compression scheme, ODETLAP. It selects a set of control points and encodes them to generate the compressed format. A full terrain can then be lossily reconstructed from these control points through an interpolation scheme that can infer local maxima. The full details of this algorithm are described by Xie, et al.\textsuperscript{8}

The main goal of ODETLAP (Overdetermined Laplacian) is to fill in gaps in the terrain, while making the terrain as smooth as possible. A overdetermined linear system is set up to solve the Laplacian equation, $z_{xx} + z_{yy} = 0$. For each grid point, the equation $z_{i,j} = (z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1})/4$ is generated. Also, for each point whose elevation is prespecified as $h_{i,j}$, the equation $z_{i,j} = h_{i,j}$ is produced. This allows these points to drift slightly from their prespecified values. This can be useful for low-precision data sets that have erroneous terracing artifacts. ODETLAP can smooth away the terracing by allowing the points to take on intermediate values. Given $k$ specified elevations on an $n$-by-$n$ grid, ODETLAP generates $n^2 + k$ equations for $n^2$ unknowns. One advantage of ODETLAP over competing interpolation schemes is that it is able to infer local maxima.

3. MULTIPLE OBSERVER SITING

Next we consider the smugglers and border guards scenario in order to evaluate our compression scheme. In this game a team places a set of observers (guards) on the terrain, and the opponent, a smuggler, must plan a path to traverse the terrain from a starting point to a specified end point. The observers are assumed to be stationary.

First we need a scheme for border guard placement. The goal is to place a set of observers on the terrain so as to maximize their joint viewshed, which is the area of the terrain that is visible by at least one observer. Due to the inherent complexity of computing the visibility between every pair of points on the terrain, it is impractical to compute the exact optimal solution to the multiple observer siting problem, especially when considering applications on small portable devices used by soldiers out in the field. Therefore, we employ the multiple observer siting algorithm developed by Franklin and Vogt,\textsuperscript{1} which uses a randomized algorithm to approximate the optimal siting of observers. There are several input parameters:

- $R$: The maximum radius of an observer’s visibility. In practice this may be limited by the transmission power of a cell tower, for example.
- $H$: The heights of the observers and targets.
- $O$: The maximum number of observers to site on the terrain.
- $A$: The minimum area to be covered by the observers.

The algorithm is divided into four major steps:

1. Vix: The visibility of each point, which is known as the visibility index, is estimated using a Monte Carlo method. For each point, $T$ points within the radius $R$ are randomly sampled. For each of the $T$ points, a line of sight is drawn to determine if that point is visible. The estimated visibility index is the fraction of the $T$ points that are visible. A typical value for $T$ is 10.
2. **Findmax**: A set of candidate points, which are known as top observers, are chosen by selecting the points with the highest estimated visibility index. To prevent these top points from being too clustered, we force them to be spread out by partitioning the terrain into blocks of size \( B \). A minimum number of points, \( P \), are chosen from each block. A typical value for \( B \) is between \( R \) and \( 2R \). We used \( B = 50 \) and \( P = 1 \) for the results below.

3. **Viewshed**: For each top observer chosen, its exact viewshed is computed, which is a bitmap of points visible by the observer. This uses a technique developed by Franklin and Ray.\(^4\) Basically, lines of sight are drawn out from the top observer in all directions. Some points may be calculated using the same line of sight.

4. **Site**: The goal is to find a subset of the top observers that covers as much of the terrain as possible. A greedy approach is taken – at each step, we choose the top observer that contributes the most to the cumulative viewshed.

![Figure 2. Siting Evaluation procedure.](image)

### 4. PATH PLANNING

After adapting multiple observer siting for border guard placement, we then developed a path planning algorithm to determine the optimal smuggler’s route. We assume that the smuggler has complete knowledge of the observers’ positions, and the smuggler would like to avoid detection while taking the quickest route possible.

The path finding routine is an adaption of the A* algorithm, and it computes the shortest path between opposite corners of the terrain while trying to avoid detection by the given set of observers. The first cost metric used was simply the number of grid points visited, ignoring the elevations and forbidding the visible regions. This method had the limitation of only minimizing the Chebyshev distance between the end points. To minimize the Euclidean distance, we implemented a two-pass system. On the first pass, all points are included in the search space, and each point is considered adjacent to its eight immediate neighbors. The result will be a path that
minimizes the Chebyshev distance. On the second pass, only points from the first path are included in the search space, and each point is considered adjacent to all other points on the path. In practice, this second pass is more efficient than the first pass. While this system is not guaranteed to minimize the Euclidean distance, it never does worse than the initial Chebyshev distance, and in practice, the Euclidean distance is usually minimized.

Figure 3. Paths that minimize Chebyshev and Euclidean distances, respectively.

We also consider a more sophisticated cost metric. A small penalty is added for traveling uphill. We also permit the smuggler to traverse an observer’s viewshed, though doing so will incur a steep penalty. The cost of moving from one point to an adjacent point is shown in equation (1), where \( h \) is the horizontal distance between the points, \( v \) is the elevation difference, \( \text{SlopePenalty} \) is \((1 + v/h)\) when going uphill and 1 otherwise. \( \text{VisibilityPenalty} \) is 100 if the new cell is visible, and \( \text{VisibilityPenalty} \) is 1 if the new cell is not visible. \((1+v/h)\) is the penalty for going uphill – If the new point is not uphill, the cost is simply \( h \times \text{VisibilityPenalty} \). Adjacent points are considered to have a horizontal distance of 1. In our data sets, for example, this is equivalent to an elevation difference of 30, so the elevations must be scaled by a factor of 1/30 for the cost metric above.

\[
\text{Cost} = \sqrt{h^2 + v^2} \times \text{SlopePenalty} \times \text{VisibilityPenalty}
\]

Table 1. Elevation ranges of the test data.

<table>
<thead>
<tr>
<th>Name</th>
<th>Elevation range</th>
<th>Elevation std dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>505</td>
<td>79</td>
</tr>
<tr>
<td>hill2</td>
<td>745</td>
<td>134</td>
</tr>
<tr>
<td>hill3</td>
<td>500</td>
<td>59</td>
</tr>
<tr>
<td>mtn1</td>
<td>1040</td>
<td>146</td>
</tr>
<tr>
<td>mtn2</td>
<td>953</td>
<td>152</td>
</tr>
<tr>
<td>mtn3</td>
<td>788</td>
<td>160</td>
</tr>
</tbody>
</table>

5. NOVEL EVALUATION METRICS

After implementing our multiple observer siting and path planning algorithms, we next formulated a policy for terrain compression evaluation. Our new test protocol for path finding is a four step algorithm. First, the original terrain representation is compressed then reconstructed to obtain the alternate representation. Second, the multiple observer siting is performed on the alternate representation to produce a set of observers, and their joint viewshed is computed. Third, this same set of observers is transferred back to the original representation, and their joint viewshed is computed on there. In the fourth step, we apply our path planning algorithm to find the optimal paths on both the original and the reconstructed representations.
We derived three new error metrics from this smugglers and border guards scenario. These error metrics are domain specific evaluation criteria for terrain compression. They target typical applications and tasks that use terrain data. If artifacts from a proposed compression scheme lead to significant errors in any of these metrics, the compression scheme should not be recommended for critical terrain applications, even if the reconstructed terrain is without visual artifacts.

The three error metrics are as follows:

1. Viewshed Error: This is the area of the symmetric difference of the cumulative viewsheds. If there are significant errors in the viewshed coverage, the siting algorithm will be unable to place the observers reliably.

2. Path Costs Error: The difference in the costs of the paths computed on the original and the alternate. If these costs are significantly different, the smuggler might plan a very inefficient path through the terrain. For example, using the alternate representation, the smuggler may take a 3-hour route, when a 1-hour route would have been discovered if he had access to the original representation.

3. Path Visibility Error: The percent of the alternate path that is visible when it is applied back to the original terrain. The smuggler plans the path using the alternate representation, believing that path to be safe. We test how much of that path actually goes through a dangerous territory. If the visibility estimate is inaccurate, the smuggler may inadvertently plan a route that traverses a guard’s field of vision.
6. RESULTS

Using our new protocol and error metrics, we compare two compression schemes: JPEG 2000 and our new approach of ODETLAP (Overdetermined Laplacian Solver). We used 400x400 terrains sampled from DTED-1 data. We’ve chosen three hilly and three mountainous datasets to standardize all our testing on, which we have named hill1, hill2, hill3, mtn1, mtn2, and mtn3.

The two schemes are competitive. There are cases where JPEG 2000 performs better, and there are cases where ODETLAP wins out. Our initial testing shows that our new approach is better when the terrain is very heterogeneous, as illustrated in Table 2.

<table>
<thead>
<tr>
<th>Original Viewshed Size</th>
<th>123416</th>
<th>125520</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate Viewshed Size</td>
<td>135998</td>
<td>137117</td>
</tr>
<tr>
<td>Viewshed Error</td>
<td>9.76%</td>
<td>9.04%</td>
</tr>
<tr>
<td>Path Cost Error</td>
<td>5.81%</td>
<td>0.23%</td>
</tr>
<tr>
<td>Path Visibility Error</td>
<td>1.56%</td>
<td>0.27%</td>
</tr>
</tbody>
</table>

Table 2. A comparison of ODETLAP and JPEG 2000.

For Table 3, we calculated each of our three error metrics under both JPEG 2000 and ODETLAP compression over six different terrains. ODETLAP is quite competitive, especially on the more mountainous terrain, such as mtn3.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Viewshed Error</th>
<th>Path Cost Error</th>
<th>Path Visibility Error</th>
<th>Viewshed Error</th>
<th>Path Cost Error</th>
<th>Path Visibility Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>0.80%</td>
<td>7.74%</td>
<td>0.12%</td>
<td>0.29%</td>
<td>0.07%</td>
<td>0.35%</td>
</tr>
<tr>
<td>hill2</td>
<td>4.93%</td>
<td>9.20%</td>
<td>1.43%</td>
<td>9.40%</td>
<td>22.76%</td>
<td>4.85%</td>
</tr>
<tr>
<td>hill3</td>
<td>0.21%</td>
<td>1.40%</td>
<td>0.00%</td>
<td>0.09%</td>
<td>4.59%</td>
<td>0.82%</td>
</tr>
<tr>
<td>mtn1</td>
<td>16.74%</td>
<td>102.70%</td>
<td>0.00%</td>
<td>23.31%</td>
<td>46.10%</td>
<td>0.00%</td>
</tr>
<tr>
<td>mtn2</td>
<td>12.70%</td>
<td>50.86%</td>
<td>1.49%</td>
<td>20.69%</td>
<td>346.91%</td>
<td>1.10%</td>
</tr>
<tr>
<td>mtn3</td>
<td>13.42%</td>
<td>183.66%</td>
<td>0.00%</td>
<td>22.62%</td>
<td>19.67%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Table 3. Our three error metrics computed on six terrains. Viewshed error is the symmetric difference between the viewsheds computed on the original and the alternate representations. Path cost error is the difference in the costs of the paths computed on the original and the alternate representations. Path visibility error is the amount of the path computed on the alternate representation that is visible in the original representation.

The visibility is usually greater on the alternate representation than on the original. This is because the compression removes detail and smooths out the terrain, eliminating visibility obstructions. The increased visibility will sometimes block off important passages, forcing the smuggler to take a long detour. This accounts for the large path cost errors, for example in mtn2 in Table 3.

The path visibility errors tend to be very small because we are sampling a portion of the terrain that is biased towards the nonvisible areas. This is a good indication that we are computing correct paths.

Our ODETLAP scheme is approaching, and in some cases surpassing, JPEG 2000. The error tends be very reasonable even with a strong level of compression. With work still ongoing, this method holds much promise.

7. AUTOMATED ROAD CONSTRUCTION

For an alternate evaluation metric, we also considered another path planning scheme unrelated to the smugglers and border guards scenario. The goal is to construct a road connecting two given points on the terrain. The slope of the final constructed road must never exceed a prescribed maximum. In order to achieve this goal, we are permitted to reshape the terrain by removing material from and depositing material onto the terrain. We
Figure 5. Hill1 dataset. For each compression technique, a set of observers are sited on the alternate representation, and their cumulative viewshed is computed. The same set of observers are then applied to the original representation, and their cumulative viewshed is computed. Then a smuggler’s path is computed separately on each representation. The two path costs are compared to see if the alternate representation plans a path that is much longer than necessary.

are not concerned with the length of the road – rather, the goal is to minimize the construction costs, which we have quantified as the amount of material displaced. We successfully adapted the A* algorithm to solve this problem.

We obtained another error metric by constructing a road separately on both the original and alternate representations of the terrain. The costs of the two roads are then compared. Table 4 shows that the errors are very small when our ODETLAP scheme is evaluated on our new metric. Examples of constructed roads are shown in Figure 11.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>% Difference in Material Displaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>0.084%</td>
</tr>
<tr>
<td>hill2</td>
<td>1.536%</td>
</tr>
<tr>
<td>hill3</td>
<td>0.093%</td>
</tr>
<tr>
<td>mtn1</td>
<td>2.054%</td>
</tr>
<tr>
<td>mtn2</td>
<td>0.004%</td>
</tr>
<tr>
<td>mtn3</td>
<td>0.034%</td>
</tr>
</tbody>
</table>

Table 4. ODETLAP, with about a 50:1 compression ratio, is evaluated on the automated road construction metric.

8. CONCLUSIONS

We met our goal of optimally siting observers on a terrain and computing good smuggler’s paths through the terrain, as well as good paths for automated road construction. We have also developed new methods for evaluating terrain compression. These are application-specific error metrics that test how well the reconstructed
terrain performs on the smugglers and border guards problem. When a user wants to plan a smugglers and border guards scenario on a reduced representation of a terrain, these new error metrics will help select the appropriate representation.

9. FUTURE WORK

Our next step is to perform more rigorous testing of our error metrics on the compression techniques. On each terrain, the path planning algorithm may be performed multiple times by selecting different start and end points. By aggregating the results from each path, we will obtain a more thorough evaluation of the terrain compression.

We will also consider alternative observer placement policies. For example, rather than seeking to maximize the total coverage area, an interesting problem would be to form a perimeter and seek to minimize the "gaps" in that perimeter.

One useful extension of the smugglers and border guards scenario is to consider mobile observers. Each guard patrols a specified path. Introducing a timing element to the smuggler’s path planning may be interesting. The smuggler may have to pause at certain intervals to wait for the guards to reposition themselves. An element of unpredictability may also be added to the guards’ movements. The smuggler will not know the observers’ future positions, though it would retain the ability to track the observers’ current positions.

10. ACKNOWLEDGEMENTS

This research was supported by NSF grants CCR-0306502 and DMS-0327634, by DARPA/DSO/GeoStar, and by CNPq - the Brazilian Council of Technological and Scientific Development.

REFERENCES

Figure 8. Mtn1 Dataset

Figure 9. Mtn2 dataset. Note that in the JPEG column, the estimated visibility is significantly greater on the alternate representation than on the original. This closes off some gaps, forcing the smuggler to take a long detour.
<table>
<thead>
<tr>
<th></th>
<th>JPEG2000</th>
<th>ODETLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Original Representation</strong></td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>Reconstructed Representation</strong></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Figure 10. Mtn3 Dataset
Figure 11. Examples of constructed roads. Note that in mtn3, the solution follows a long contour in the terrain.
ABSTRACT
We present the GeoStar project at RPI, which researches various terrain (i.e., elevation) representations and operations thereon. This work is motivated by the large amounts of hi-res data now available. The purpose of each representation is to losslessly compress terrain while maintaining important properties. Our ODETLAP representation generalizes a Laplacian partial differential equation by using two inconsistent equations for each known point in the grid, as well as one equation for each unknown point. The surface is reconstructed from a carefully-chosen small set of known points. Our second representation segments the terrain into a set of regions, each of which is simply described. Our third representation has the most long term potential: scooping, which forms the terrain by emulating surface water erosion.

Siting hundreds of observers, such as border guards, so that their viewsheds jointly cover the maximum terrain is our first operation. This process allows both observer and target to be above the local terrain, and the observer to have a finite radius of interest. Planning a path so that a smuggler may get from point A to point B while maximally avoiding the border guards is our second operation. The path metric includes path length, distance traveled uphill, and amount of time visible to a guard.

The quality of our representations is determined, not only by their RMS elevation error, but by how accurately they support these operations.

Categories and Subject Descriptors
E.2 [Data Storage Representations]: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.4.2 [Image Processing and Computer Vision]: Compression

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

Introduction
The increasing amounts of terrain data, from Light Detection and Ranging (LiDAR) and Interferometric Synthetic Aperture Radar (IFSAR), such as from the 2000 Shuttle Radar Topography Mission (SRTM), create both an opportunity and a problem. The former is the set of new operations that can be performed, while the latter is the difficulty of storing the data and efficiently processing it. Many earlier “toy” algorithms have asymptotic times that grow too quickly with data size to remain useful. Paradoxically, as computers get faster and memory sizes get bigger, efficiency can become more important. In the authors’ view, because hard disk capacities are growing faster than their speeds, the advantage of using only primary storage (a.k.a. internal memory or RAM), when feasible, compared to using external memory also grows.

In this context, terrain means elevation above some geoid (the assumed sea level, extrapolated over land). We do not consider the important geopotential issues of defining and determining geoids.

The subject of this paper is Smugglers and Border Guards, the GeoStar project at RPI. This paper describes the whole system and presents newer results. We are researching alternate terrain representations, and terrain operations. The operations are then used to evaluate the representations.

As the project has progressed, various representations have been studied, ranging from new twists on classics, to more mature representations with immediate potential, like ODETLAP, through to radical ideas with both great potential and great difficulties, like scooping. The major operations being researched include multiple observer siting, and path planning to avoid the observers. Longer term ideas here include allowing earth moving operations on the paths. RPI is one of several teams funded under the GeoStar program by the Defense Science Office of DARPA.

from contours.

The classic Triangulated Irregular Network, a piecewise linear, non tensor product, spline, was first implemented in GIS by Franklin[5]. Later extensions include Speckmann and Snoeyink[29], who process very large sets of irregular points externally. Lavery[23] and Zhang[31] have researched higher order, non tensor product, splines, which do not have extraneous oscillations. Most of the works on concise representation of the terrain data concentrate on efficient representation and manipulation of TINs. Samples of those works are in Park[22, 27], Kim[21], and Iseburg[20].

Yet another series of works target multiresolution representation of terrain elevations. Hoppe develops a progressive mesh transmission method based on edge collapse and feature retention in [19]. Related works can be found in Garland[13] and deFloriani[4]. In a different work, Losasso and Hoppe[24] model terrain elevation grids retaining different levels of detail, which are then compressed and progressively transmitted based on the field of view of the user. De Floriani and Magillo[3] compare different Multiresolution TIN proposals. Among multiresolution methods, wavelet based schemes are also notable. The SPIHT[28] and JPEG-2000[26] methods for multiresolution compression are patented, and hence less useful.

There has also been much research in the Computational Geometry community into surface reconstruction, e.g., of 3D objects from point clouds, Dey[2]. Reconstructing terrain is somewhat different since terrain is a single-valued function whose topology is known, and whose known points may be very unevenly spaced.

2. ALTERNATE TERRAIN REPRESENTATIONS

The first goal of the RPI GeoStar project is to produce alternate terrain representations that take less space, but are lossy. We are pursuing several representations in parallel, with a goal of both short term results and long term potential. Of our major representations, ODETLAP is the most mature, segmentation is a work in progress, and scooping has the most potential.

2.1 Terrain Properties

Ideally, some formal terrain model would guide any evaluation of terrain representation. Since that does not yet exist, we can only study actual terrain. The challenge is that any mathematical representation of terrain must have a goal to acknowledge its properties, such as the following.

- Real terrain is more irregular than databases such as DEM-1 cells. Algorithms tested only on that data might unknowingly be exploiting their artificial smoothness. - Terrain is not differentiable many times, i.e., it is generally not \( C^n \) for \( n > 0 \). Indeed, the physical phenomena that generate terrain generally do not depend on, or generate, high order continuity. The major exception is the curvature, in the horizontal plane, of stream beds. - In places, the terrain is \( C^{-1} \), i.e., discontinuous. Indeed, although techniques such as contour lines have difficulty representing them, these may be the most important features for many users. Discontinuities strongly affect both visibility and mobility. - The data is heterogeneous; different regions have different statistics. For example, river basins occur mostly above sea level, while mid-ocean ridges occur under sea level. Some regions above sea level are karst terrain, with sink holes, while other regions have rivers.

- The heterogeneity gets worse if we consider other planetary bodies, such as the Moon, because of the varied formation mechanisms, such as impact craters or large volcanoes.

- There are long range correlations, such as river basins, that may extend from one side of a continent almost to the other ocean.

- Terrain is often not spatially symmetric in the horizontal direction. Rivers’ headwaters, such as the Amazon’s, are often near the opposite edge of the continent from their deltas.

2.2 Nonlinearity is Powerful

This research is biased towards nonlinear, instead of linear, numerical techniques. Nonlinearity is a very powerful, albeit hard to use, approximation technique. Even for \( C^n \) quickly convergent functions like \( \exp(x) \), the best rational approximation is more efficient than the best polynomial one, Newman[25]. (Note that the Taylor expansion is far from the best polynomial approximation; a Chebyshev is almost optimal.) However the true power is revealed when approximating functions like \( \abs(x) \) or a step function. Because they are \( C^n \) and \( C^{-1} \) respectively, uniform polynomial approximations do not exist. Note that the best rational approximation is more than just a Padé approximation, which is properly defined as a formal transformation from a polynomial, ignoring convergence.

2.3 ODETLAP

2.3.1 Definition

ODETLAP is an algorithm and implementation for

1. selecting a set of points that characterize a terrain elevation array, and

2. reconstructing a terrain elevation array from a set of points.

The process is summarized in Figure 1. Currently we use \( 400 \times 400 \) arrays for implementation convenience in Matlab; larger arrays are possible with better matrix algorithms. The purpose of the ODETLAP point selection is that the resulting set of points, perhaps 1000 in number, characterize the surface well, and can be stored in much less space than...
the original array. The purpose of the ODETLAP reconstruction is to produce an array of elevations from a small point set. The reconstruction can process any set of points. For example it is also useful to fit a surface to a set of contours (the original application). The points may even be inconsistent; then a best fit will be computed.

2.3.2 Properties

We originally developed ODETLAP to address a shortcoming in some algorithms for filling in elevation contours to produce a matrix of elevations. That problem is that the original contours are too often visible in the generated surface. While we know little about formally modeling terrain (see section 2.1 above), we do consider it extremely unlikely that the real terrain is terraced at exact multiples of 10m.

ODETLAP also has many other advantages, which are generally not shared by competing surface approximation methods.

• It can handle continuous contour lines of elevations, w/o needing to select only a subset of those points for processing.
• It can handle kidney-bean-shaped contours w/o generating fictitious flat regions inside (as happens with interpolation methods that run straight lines out from the unknown point to the closest contour in each direction).
• It can handle broken contour lines (unlike methods running straight lines until they hit a contour).
• It can handle isolated points.
• It can infer, from a set of concentric contours, a mountain top (local maximum) that is higher than the highest contour.
• It can handle very unevenly distributed data.
• It can confute inconsistent data, say a small hi-precision region overlaid on a large lo-precision region.
• It enforces continuity of slope across contours, so that they do not show in the resulting surface, i.e., no generated terraces.

How well ODETLAP works is shown in Figure 2, where the input, designed to be nasty, is several nested square contours, whose \( C^0 \) continuity at the corners should be challenging for any algorithm. The silhouette edge of the fitted surface shows almost no evidence of the contours. The max absolute error is 12%, and the mean error 2.7%, of the elevation range.

2.3.3 Algorithm

ODETLAP stands for Overdetermined Laplacian Partial Differential Equation. The ODETLAP representation consists of

1. \( \mathcal{P} \), a set of important points on the surface. \( \mathcal{P} \) is coded to minimize the number of bits needed to store it.
2. The ODETLAP algorithm for reconstructing the elevation matrix from \( \mathcal{P} \).

ODETLAP is lossy, and allows a tradeoff between representation size and elevation error. ODETLAP allows for the terrain to be progressively transmitted. Since the points \( p \in \mathcal{P} \) are ordered by importance, they may be transmitted one-by-one. As each \( p \) is received, the receiver may reconstruct terrain \( T_i \) and evaluate it. If \( T_i \) is good enough, then the receiver tells the transmitter to stop. This is useful because there are applications where bandwidth, storage, and power consumption are still critical.

2.3.4 Point Selection

How should we select the points? Extensive experiments have shown that the reconstruction algorithm is surprisingly robust. Therefore the following greedy algorithm suffices.

• Select \( \mathcal{P}_0 \), an initial set of points by a method such as using TIN incrementally to insert 100 points. • Reconstruct the DEM determined by those points. • Compute the error matrix between the original terrain and the above solution. • Select \( \mathcal{A}_i \), the set of 10 points with the largest error. (We investigated other strategies, but those were all worse.) • Insert them into \( \mathcal{P}_i = \mathcal{P}_{i-1} \cup \mathcal{A}_i \). • Repeat until the error is small enough.

2.3.5 Point Coding

Since our goal is to represent the surface as compactly as possible, coding the points to require the fewest number of bytes is as important as choosing the smallest set of points. Indeed, it might be best to select more points that can be coded into fewer bytes (because they form a regular pattern). We investigated several point coding strategies, and currently prefer the following method:

1. Represent the \((x,y)\) as 1-bits in a 400 \times 400 bitmap image that whose bits are otherwise 0.
2. Consider the image as a string of 160000 bits that has occasional 1s separated by strings of 0s. If two 1s are adjacent, then consider that the intermediate string of 0s has length zero.
3. This bit string may be represented as a list of the lengths of the zero strings (i.e., run-length encoding).
4. Use some efficient method to encode the lengths, noting that most of them are < 254 but some are larger.
5. Code the sequence of \( z \) by taking deltas and then using bzip2.

2.3.6 Surface Reconstruction

ODETLAP reconstructs the matrix \( z_{ij} \) of terrain elevations from \( \mathcal{P} \), a scattered set of elevations, with an extension of a Laplacian partial differential equation (PDE), also known as a heat flow equation. A Laplacian PDE is solved by defining a sparse system of linear equations.

\[
4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} \quad (1)
\]

for every unknown non-border point. Border points are a complicated special case to be discussed elsewhere. Unfortunately, the solution has several bad properties. • There is no information flow across contours; therefore the reconstructed surface is not \( C^2 \) there, and the contours are very
the terrain inside the innermost contour around a mountain range of the known datapoints that they are derived from, cloth.

- Between contours, the surface sags like a piece of cloth. Since interpolated values can never be outside the range of the known datapoints that they are derived from, the terrain inside the innermost contour around a mountain top is flat like a mesa. To remedy that, we extend the Laplacian PDE as follows.

1. Define equation (1) above for every non-border point, whether known or unknown.

2. For each point with a known elevation \( h_{ij} \), define an additional equation \( z_{ij} = h_{ij} \).

Since the known points have two inconsistent equations, the system is now overdetermined. We solve for a best fit for our desired relative weights for the two different equation types, Cousie[15, 16, 17]. The resulting surface does not exactly fit the known points, which, given the (lack of) accuracy of real data, is an advantage. An alternative would be to consider the surface as a thin plate, and minimize its bending energy. The PDE is \( z_{ij}^{2} + 2z_{ij} + z_{ij} = 0 \). However, this causes a ringing or Gibbs phenomenon, and adds extra complexity w/o a corresponding extra benefit.

ODETLAP’s novelty is the overdetermined system, which was not feasible until recent large sparse system solution techniques were developed. Any resemblance to interpolation with springs is only superficial.

2.3.7 Results and Status

We used six test terrains, three hilly and three more mountainous, shown in Figure 3. Table 1 shows the results when 1000 points were used. All sizes are in bytes. The \( XY \) Size is the size of the \( (x, y) \) coordinates of the points, when run-length coded. The \( Z \) Size is the size of the \( z \) coordinates, delta encoded and bzipped. The Total Size is their sum. That is the size of the terrain in our alternate representation. The original size of each 400 x 400 terrain, at 2 bytes per point, is 320KB, and their ratio is the compression ratio of our representation. Since our representation is lossy, there is the usual size-accuracy tradeoff. Therefore, we give the RMS elevation error of our representation, and the elevation range of the data, and the ratio of the RMS error to the range, as a percentage.

Our current work includes optimizing ODETLAP by exploring various point selection techniques, better coordinate coding techniques, and different PDEs. We know how to run ODETLAP on larger datasets, by using a Page-Saunders algorithm as used by Childs[1], but are staying with 400 x 400 arrays for the moment since they are faster and easier to process.

2.4 Terrain Segmentation

Inanc[8] has shown that encoding terrain elevation data through segmentation is an enabling method for a lossy compression. In this work we propose some extensions, which can allow better plane compression and provide a lower average error.

The input to our problem is an elevation dataset \( T \) consisting of \( N \times N \) elevation postings. Ideally each elevation posting is an \( (x, y, z) \) triplet but since we are dealing with a regular grid, the \( (x, y) \) values are implicit and only the \( z \) values are stored. Thus we can conveniently store \( T \) in a matrix of size \( N \times N \). A further simplification is that DEMs often store their elevation values as 16-bit integers.

We attack the problem of finding the best fitting 2D manifolds by generating a set of candidates. Candidates are generated from small local terrain patches. One way is to partition our terrain into square tiles of size \( t_{x} \times t_{y} \). To capture minute variations in the terrain we pick \( t_{x} = 2 \), thus generating \( N^{2}/4 \) tiles. Each tile contains \( t_{z}^{2} \) elevations \( (z) \) values, which are modeled by the following linear system: \( Xc = z + e \). where predictor variables \( X \) are the implicit grid coordinates \( (x, y) \) and a constant factor. The fitting error is \( e \).

A multiple parameter linear regression function solves for the coefficient vector \( c \) of the best fitting plane. Those coefficients are stored in a list \( L \) for future consideration. For each entry in the list \( L \), the plane is extrapolated from the small tile it originates to, the entire terrain. The \( (x, y) \) coordinates of the entire terrain \( T \), together with a constant factor make the matrix \( X_{t} \). The process generates a 2D manifold \( \tilde{z} \), which is a crude approximation for the entire terrain: \( \tilde{z} = X_{t}c \). The fitness of the approximation \( \tilde{z} \) is tested using the infinity norm: \( ||z - T||_{\infty} \). We would like to limit this value to a user specified constant \( \gamma \) we call loss factor: \( LF \).

A way of doing that is to limit the 2D manifold \( \tilde{z} \) to a set of \((x, y)\) coordinates, which meet the constraint. We call the size limited \( \tilde{z} \) a segment: \( S \). A segment \( S \) consists of a manifold \( \tilde{z} \) and a set of \((x, y)\) coordinates on that manifold. Our model depends on a set of segments, which cover all \((x, y)\) coordinates in \( T \).

2.4.1 Segment Selection

After our 2D manifold generation scheme populates the list \( L \) with candidates, we need to pick a minimal set that will cover all \((x, y)\). Compression is achieved since a single segment may contain a large number of elevation postings, which are all concisely modeled. The obvious algorithm is the greedy heuristic, where at each step we pick the largest contributing segment and we stop when the coverage is achieved.
<table>
<thead>
<tr>
<th>Data</th>
<th>XY</th>
<th>Z</th>
<th>Total</th>
<th>Orig</th>
<th>Compr.</th>
<th>RMS</th>
<th>Elev</th>
<th>Err</th>
<th>Ratio</th>
<th>Err</th>
<th>Range</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1</td>
<td>1250</td>
<td>1304</td>
<td>2554</td>
<td>320K</td>
<td>125.</td>
<td>3.62</td>
<td>505</td>
<td>0.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hill2</td>
<td>1243</td>
<td>1354</td>
<td>2597</td>
<td>320K</td>
<td>123.</td>
<td>9.45</td>
<td>745</td>
<td>1.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hill3</td>
<td>1279</td>
<td>1209</td>
<td>2488</td>
<td>320K</td>
<td>129.</td>
<td>1.72</td>
<td>500</td>
<td>3.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mtn1</td>
<td>1228</td>
<td>1456</td>
<td>2684</td>
<td>320K</td>
<td>119.</td>
<td>17.34</td>
<td>1040</td>
<td>1.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mtn2</td>
<td>1244</td>
<td>1424</td>
<td>2668</td>
<td>320K</td>
<td>120.</td>
<td>17.17</td>
<td>953</td>
<td>1.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mtn3</td>
<td>1241</td>
<td>1503</td>
<td>2744</td>
<td>320K</td>
<td>117.</td>
<td>17.06</td>
<td>788</td>
<td>2.2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: ODETLAP Results

![Figure 4: Seven 10m datasets, colormaps are not one-to-one.](image)

<table>
<thead>
<tr>
<th>LF</th>
<th>LF</th>
<th>LF</th>
<th>Low</th>
<th>High</th>
<th>Elev.</th>
<th>Elev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>20</td>
<td>833</td>
<td>833</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>50</td>
<td>55</td>
<td>119</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>65</td>
<td>52</td>
<td>35</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>28</td>
<td>59</td>
<td>85</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>59</td>
<td>59</td>
<td>31</td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>119</td>
<td>119</td>
<td>33</td>
<td>33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Number of segments, elevation extremes.

### 2.4.2 Results

We present results on seven different terrain elevation datasets. Those are 400 × 400 size datasets with horizontal resolution of 10m, from the 10m USGS DEMs covering Hawaii[14]. We used datasets containing different geological features (e.g., mountains, valleys, plateaus, hills, plains, cliffs), Figure 4. We try three different LF (loss factor) values of 5, 10 and 20. As expected the number of segments drops the higher the LF. This trend can be observed in Table 2. We also report the lowest and the highest elevation of the dataset on the same table.

For each segment we need to encode the plane coefficients and (x, y). We combine segments in a single indexmap and apply entropy coding with the PPMII encoder from the LEDA library. The resultant compressed size is in Table 3. We compare those results to gzip compression of the datasets. We again get a significantly smaller footprint at the price of a controlled data loss.

We expect the worst reconstruction to be the one using the least number of segments. In our case this is lanai-south at LF = 20, Figure 5.

**2.5 Scooping**

Imagine starting with a high plateau and carving the terrain with multiple passes of a giant shovel. For each pass, we insert the shovel at some point and then dig in a continuous motion towards the edge. As we do that, we may keep the depth of the blade level, or push it deeper into the earth, but we never make it shallower.

![Figure 5: Lanai-south original and restored after compression using only 9 planes.](image)
Scooping has several properties. • It will not create a local minimum. This desirable feature contrasts to every other known terrain representation method. • It naturally lends itself to the creation of complex drainage systems, again in contrast to other representations. • It is quite nonlinear, and so has a power not available to linear methods. • Slope discontinuities and cliffs can be created as desired. • The complete series of scoops representing a cell may be truncated at any point to produce a less accurate representation of that cell that still looks like terrain. Therefore the terrain may be lossily compressed and progressively transmitted.

None of the above properties pertain to a Fourier expansion. Scooping may also be visualized as a machining operation with a 3-axis drilling machine, if we assume that each pass of a drill extends to the edge of the workpiece, and the drill’s depth never decreases during the pass.

More formally, this, our far-reaching proposed approach is to develop new mathematical morphological operators to enable parsimonious and compact representations of terrain, such as a scooping operator for representing terrain elevation. The uniqueness of this idea is to lay a formal foundation for terrain, to allow a formal inquiry into the best algorithms for applications, such as compression, visibility, mobility, drainage, the representation of multiple related data layers, and multiple data source conflation. This will improve on current methods of testing heuristics on test samples.

What terrain operators are appropriate, and how realistic should they be? While Fourier series are too unrealistic, a complete geological evolution model is too complex. Our scooping operator, analogous to scooping earth out of the side of a hill, will initially proceed as follows.

Although scooping has the greatest longterm potential, it is also the most difficult to research, and so we have no concrete results to report yet.

3. OPERATIONS ON TERRAIN

We researched and implemented two major terrain operations: multiple observer (“border guards”) siting, and (“smuggle’s”) path planning to avoid the observers.

3.1 Multiple Observer (Border Guard) Siting

Where should we site a set of observers, such as border guards, so every point on the terrain (or more likely, 90% of the points) can be seen by at least one observer? The goal is either to minimize the number of observers needed to cover a specific fraction of the terrain, or to maximize the amount of terrain covered by a given fixed number of observers. This process has various parameters, such as the observer and target height above the local terrain, and the radius of interest, the distance out to which each observer can see. A variant of the siting problem is to enforce intervisibility, requiring that enough observers can see each other that they form a connected graph, enabling observers to communicate with each other, perhaps indirectly. This research theme goes beyond the theme of more accurately computed viewsheds of single observers. We have a siting testbed, capable of easily processing level-1 DEMs. Figure 6 shows sample output with and without intervisibility being enforced. The terrain is the USGS Lake Champlain West cell. The details of this method are omitted since they have been reported earlier, in Franklin[7, 10, 11, 12].

Figure 6: Multiple Observer Siting with and w/o Intervisibility

3.2 Smugglers’ Path Planning

How should a smuggler travel to minimize his time visible to the optimally sited border guards?

The path finding routine implements the A* algorithm and computes the shortest path between opposite corners of the terrain while trying to avoid detection by the given set of observers. In the algorithm, the cost of moving from one point an adjacent point uphill is \( \sqrt{h^2 + v^2} \cdot (1 + v/h) \cdot P \), where \( h \) is the horizontal distance between the points, \( v \) is the elevation difference, \( P \) is a Visibility Penalty, chosen to be \( P = 100 \) if the new cell is visible, or \( P = 1 \) if the new cell is not visible. If the new point is not uphill, the cost is simply \( h \times P \).

The path-finder is a two-pass system. On the first pass, all points are included in the search space, and each point is considered adjacent to its eight immediate neighbors. The result will be an approximately minimal path. On the second path, only points from the first path are included in the search space, and each point is considered adjacent to all other points on the path. The result will be a path that more nearly minimizes the Euclidean distance. In practice, this second pass is very efficient.

4. EVALUATION

In addition to computing the RMS error for the alternate representations, as a function of that representation’s size, we sought a more sophisticated evaluation procedure, summarized in Figure 7. The goal is to answer the question of whether our alternate representations are suitable for sophisticated operations. That is, how good is a siting or path planning operation that is performed on terrain compressed with, say, ODETLAP? Designing an appropriate metric takes care. For instance, a small change in the terrain may cause a large change in a computed path, if several possible paths have approximately the same cost. What is important is whether the path computed on the alternate representation has the same cost as the path computed on the original representation.

Since the siting and path planning has many parameters, our evaluation is still preliminary. However, Table 4 has some indicative results. It evaluates one compressed terrain dataset on three metrics of increasing complexity.

1. Viewshed Error: A set of observers is sited on both the original and alternate representations, the two cumulative viewsheds computed, and the area of their symmetric difference reported.
2. **Path Length Error:** A smuggler’s path is computed to avoid the observers in the two cases, and the difference in their lengths is reported.

3. **Path Visibility Error:** The path computed on the alternate terrain is transferred to the original representation and tested against the original viewsheds. The percent of that path that is now visible is reported.

<table>
<thead>
<tr>
<th>Viewshed Error</th>
<th>JPEG 2000</th>
<th>ODETLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path Length Error</td>
<td>5.81%</td>
<td>0.23%</td>
</tr>
<tr>
<td>Path Visibility Error</td>
<td>1.56%</td>
<td>0.27%</td>
</tr>
</tbody>
</table>

Table 4: Viewshed and Path Planning Evaluation of ODETLAP Terrain Compression

In each case, smaller numbers are better. We also performed these tests on the terrain compressed with JPEG 2000 to about the same size. The two schemes are competitive. There are cases where JPEG 2000 performs better on the less sophisticated metrics. However ODETLAP is much better on path planning. It also appears that our new approach is better when the terrain is very heterogeneous. ODETLAP also has the many other advantages listed earlier, which JPEG-2000 lacks. Also, we are still improving ODETLAP.

5. **FUTURE**

This is a work in progress with many open possibilities, such as scooping. For ODETLAP, we are investigating different point coding techniques and hierarchical extensions.

For terrain segmentation, we can imagine that there might be a spike or a well in the elevation data, which might not belong to either of the segments. To model these aberrations, we stipulate that certain points will be stored separately. This mechanism can also be used to store survey points, which have higher accuracy than the rest of the data.

For path planning, we are allowing earthmoving operations, and wish to minimize a sophisticated cost function, while respecting physical rules such as the maximum slope of the road under construction.

Finally, there is a great potential for end-to-end optimization of the representations and operations as parts of one unit, producing more compact representations supporting more powerful operations.

6. **ACKNOWLEDGMENTS**

This research was supported by NSF grants CCR-0306502 and DMS-0327634, by DARPA/DSO/GeoStar, and by CNPq-the Brazilian Council of Technological and Scientific Development.

7. **ADDITIONAL AUTHORS**

Additional authors: Frank Luk, CS Dept, RPI.

8. **REFERENCES**

Smugglers and Border Guards: the Geo* Project at RPI

Presented at
The 15th ACM International Symposium on Advances in Geographic Information Systems in 2007
(ACM GIS 2007)
Seattle
Nov 8 2007
http://www.cise.ufl.edu/dept/acmgis2007/

W Randolph Franklin
Rensselaer Polytechnic Institute
Troy NY 12180
http://wrfranklin.org/

A Terrain Representation

Not produced by a sponsor of this conference.
Smuggling and Border Guards: the Geo* Project at RPI

- ODETLAP – Very compact terrain representation,
- Border guards - Multiple observer siting to maximize joint viewshed,
- Smugglers - Path planning to avoid viewsheds
ODETLAP Process

First ODETLAP mention: my 1998 Spatial Data Handling Symposium (Vancouver) talk.

ODETLAP Point Selection

Several options:

- Incremental TIN to find most important points, then greedy insertion of worst points (Allows progressive transmission)
- Regular grid of points (more points, but compress better) (More compact)
- Stream and ridgeline points (Preliminary)
Incremental Triangulated Irregular Network (TIN)

- Can process 10^8 points on a laptop.
- Works in memory w/o needing to page data from disk.
- Inserts points incrementally, in order of importance.
- Can progressively transmit terrain.
- Identifies ridge lines automatically.

Coding the Points to Reduce Space

- As important as point selection.
- Code (x, y) separately from z.
- If (x, y) a regular grid: give its resolution
- Else: run-length encode the bitmap.
  - 0100000011000010001 -> 16043
  - Only about 1000 of 160,000 bits are 1.
- Only 20% over info theoretic min:
  \[ \lg(\text{choose}(160000, 1000)) = 8754 \text{ bits} \]
- Z: delta code, then gzip2.
  - 100 125 90 90 100 -> 100 25 -35 0 10

- Zhongyi Xie
Traveling Salesman Path

- **Hypothesis:** nearby points often have nearby Z, which delta code better
- Find a traveling salesman path through the selected points.
- Put the Z in that order and code them.
- (X,Y) coding is not affected.
- **Status:** have some preliminary results.

- *Barb Cutler*

Info theoretic min for (x,y)

- Assume that 1000 of 400x400 bits are 1, rest are 0.
- That’s why we separate (x,y) from (z).
**ODETLAP Reconstruction - 1**

- Extension of classical Laplacian partial differential equation used to solve heat flow etc
- Now possible with new numerical computation techniques on large sparse overdetermined systems of linear equations
- Adds capabilities to the classical PDE
  - Local maxima inference
  - Inconsistent data conflation

**ODETLAP Reconstruction - 2**

- Solve overdetermined variant of Laplacian PDE.
  - Known pts: $z_{ij} = h_{ij}$
  - All pts: $4z_{ij} = z_{i-1j} + z_{i+1j} + z_{ij-1} + z_{ij+1}$
- Easily processes 400x400 arrays of elevation posts in Matlab (160,000 unknowns)
- Process larger arrays with Page-Saunders technique (done by John Childs in 2003)
Four Matlab Interpolation Techniques on Nested Square Contours

This difficult example was chosen to illustrate all these methods’ limitations.

ODETLAP on Nested Squares

Surface now looks much better. Can tradeoff accuracy vs smoothness.
Terrain Test Data

Extracted from level 2 DEMs

Elevation range

TIN + Greedy ODETLAP Results

<table>
<thead>
<tr>
<th>Data</th>
<th>Size, bytes</th>
<th>Compression ratio</th>
<th>RMS Elev Error, m</th>
<th>RMS Slope Error, deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>1880</td>
<td>170:1</td>
<td>2.83</td>
<td>3.53</td>
</tr>
<tr>
<td>hill2</td>
<td>1962</td>
<td>163:1</td>
<td>4.06</td>
<td>8.06</td>
</tr>
<tr>
<td>hill3</td>
<td>1739</td>
<td>184:1</td>
<td>1.66</td>
<td>1.65</td>
</tr>
<tr>
<td>mtn1</td>
<td>1979</td>
<td>162:1</td>
<td>3.77</td>
<td>14.0</td>
</tr>
<tr>
<td>mtn2</td>
<td>2006</td>
<td>160:1</td>
<td>4.31</td>
<td>14.1</td>
</tr>
<tr>
<td>mtn3</td>
<td>2004</td>
<td>160:1</td>
<td>4.58</td>
<td>13.3</td>
</tr>
</tbody>
</table>
TIN+Greedy Elevation Accuracy

Compressed Size vs. Error

% RMS Elevation Error/Range

Compressed Size (Bytes)

Hill 1
Hill 2
Hill 3
Mtn 1
Mtn 2
Mtn 3

TIN+Greedy Slope Accuracy

Compressed Size vs. Error

RMS Slope Error (degrees)

Compressed Size (Bytes)
Different Point Selection Strategies

- **Previous slides**: TIN+greedy.
- Allows progressive transmission of the points, *(if replace bitmap coding of the (X,Y) with a bzip2 compression)*.
- **Following slides**: regular grid point selection.
- Compresses better, but no progressive transmission.

- *Marcus Andrade*

Regular Grid ODETLAP Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Points</th>
<th>Compressed Size</th>
<th>Compression Ratio</th>
<th>Elev RMS (m)</th>
<th>Slope RMS (deg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1</td>
<td>529</td>
<td>306</td>
<td>1046:1</td>
<td>9.63</td>
<td>4.32</td>
</tr>
<tr>
<td>Hill2</td>
<td>1600</td>
<td>807</td>
<td>397:1</td>
<td>9.98</td>
<td>6.54</td>
</tr>
<tr>
<td>Hill3</td>
<td>225</td>
<td>172</td>
<td>1860:1</td>
<td>9.71</td>
<td>3.04</td>
</tr>
<tr>
<td>Mtn1</td>
<td>4489</td>
<td>2194</td>
<td>146:1</td>
<td>9.66</td>
<td>10.13</td>
</tr>
<tr>
<td>Mtn2</td>
<td>4489</td>
<td>2027</td>
<td>158:1</td>
<td>9.95</td>
<td>10.34</td>
</tr>
<tr>
<td>Mtn3</td>
<td>4489</td>
<td>2013</td>
<td>159:1</td>
<td>9.91</td>
<td>9.85</td>
</tr>
</tbody>
</table>

(Our second ODETLAP point insertion strategy)
Regular Grid ODETLAP Accuracy

Missing Data Fillin

ODETLAP

Laplacian

Thin plate

Matlab nearest

Matlab linear

Matlab cubic

Metin

Inanc

RPI Geo* / ACM-GIS Nov 2007
Goal 2: Smugglers and Border Guards (aka Siting & Path Planning)

Parameters:
- Observer height
- Target height
- Radius of interest
- Intervisibility?

Siting program

Observer positions

Joint viewshed

Multiobserver sitting steps

1. Compute approximate visibility index of every possible observer.
2. Compute exact viewsheds of the best.
3. Greedily insert potential observers into the final set of observers, maintaining a bitmap of the cumulative viewshed.
4. Intervisibility: insert only visible observers.

**Key:** fast bitmap operations allow hundreds of observers to be sited with hi-res viewsheds.
Sample Viewsheds

Note the level of detail

Siting Toolkit by ESRI

- **ArcGIS DLL** toolkit: an operational class configurable to perform siting simulations on any platform with a C++ compiler.
- Includes an ArcMap command application on Windows, to demonstrate its capabilities.
- Both are functional and scalable.
- Marquee Tool:
**ArcGIS DLL Dialog Box**

![ArcGIS DLL Dialog Box Image]

**Path Planning (Smugglers)**

- Find cheapest path between source and goal.
- Cost metric is not simply path length:

\[
\begin{align*}
\mathbf{c} &= \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2} \\
&\quad \cdot \left( 1 + \max \left( 0, \frac{\Delta z}{\sqrt{(\Delta x)^2 + (\Delta y)^2}} \right) \right) \\
&\quad \cdot (1 + 100\nu)
\end{align*}
\]

(\text{Distance})

(\text{Climbing costs})

(\text{BIG penalty for being seen})
Path planning algorithm

- Designed for hi-res, say 1000x1000, maps.
- Impossible to form the $10^6 \times 10^6$ cost matrix.
- Use A* to search for initial feasible, good, path.
- Iterate to optimize it.
- Doesn’t hang up on local optima.
- Compute many paths to evaluate compression throughout the terrain.
- Note how complex our paths are.
- Video: multipath.wmv - Dan

Many Paths on Each Dataset

![Image of many paths on different datasets]
Smugglers Path Evaluation of ODETLAP

- **Size**: size of compressed dataset in bytes. Original binary size=320KB.
- **Incr. cost**: extra cost of optimal path computed on compressed dataset and evaluated on original dataset.

<table>
<thead>
<tr>
<th>Data</th>
<th>Size</th>
<th>Compr. Ratio</th>
<th>Incr. Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>1763</td>
<td>182:1</td>
<td>5.5%</td>
</tr>
<tr>
<td>hill2</td>
<td>1819</td>
<td>176:1</td>
<td>6.1%</td>
</tr>
<tr>
<td>hill3</td>
<td>1607</td>
<td>199:1</td>
<td>4.4%</td>
</tr>
<tr>
<td>mtn1</td>
<td>1925</td>
<td>166:1</td>
<td>19.2%</td>
</tr>
<tr>
<td>mtn2</td>
<td>1884</td>
<td>170:1</td>
<td>18.2%</td>
</tr>
<tr>
<td>mtn3</td>
<td>1946</td>
<td>164:1</td>
<td>17.0%</td>
</tr>
</tbody>
</table>

Path Planning for Road Construction

- **Goal**: Construct an optimal road connecting two points.
- **Allowed**: Material removal and deposition.
- **Constraint**: Max allowable slope is bounded.
- **Objective function**: Amount of material moved.
- **Method**: A*
Before merging                      After merging

Larger more realistic watersheds and drainage networks, feed points into ODETLAP  
- Jon Mucke

Future - Scooping

- Terrain not continuous.
- *Thesis*: methods (Fourier, ODETLAP, ...) assuming that are wrong.
- Model terrain as sequence of terraforming operators.
  - Water erosion →
- Implement descriptive geography
- Easy to say, hard to do.
Team

- Prof Randolph Franklin – helping everyone
- Prof Barbara Cutter – computer graphics
- Prof Frank Luk (*on leave as Vice-President (Academic) of Hong Kong Baptist U*) – numerical analysis
- Prof Marcus Andrade – visiting from UF Viçosa (Brazil) – computational geometry
- Metin Inanc – ODETLAP
- Zhongyi Xie – ODETLAP
- Dan Tracy – multiobserver siting, path planning
- Jon Muckell – hydrology
- NSF, DARPA – financial support, direction.

Summary

- Represent terrain in 1% of original binary space with compression ratios of 80:1 to 500:1 with 10m elevation and 5-10 degree slope error.
- Site border guards, plot smugglers paths.

Future

- Scooping.
- Hydrology.
- Better slope representation.
- Large urban datasets.
Drainage Network and Watershed Reconstruction on Simplified Terrain

Jonathan Muckell\textsuperscript{a}, Marcus Andrade\textsuperscript{b}, W. Randolph Franklin\textsuperscript{a}, Barbara Cutler\textsuperscript{c}, Metin Inanc\textsuperscript{c}, Zhongyi Xie\textsuperscript{c} and Daniel M. Tracy\textsuperscript{c}

\textsuperscript{a} Department of Electrical, Computer, & Systems Engineering, Rensselaer Polytechnic Institute
\textsuperscript{b} Universidade Federal de Viço\~{s}a, Viço\~{s}a, Brazil
\textsuperscript{c} Department of Computer Science, Rensselaer Polytechnic Institute

Abstract

We present a new form of terrain compression to preserve the hydrological information that is lost using standard terrain simplification techniques. First, we compute the drainage by using a system of linear equations to determine the amount of water flowing into each cell. The flow is then computed on the inverted terrain which provides an approximation of the ridge network. The drainage and ridge networks are simplified using the Douglas-Peucker algorithm, selecting the most significant points. These points represent our compressed version of the hydrology. To uncompress, we use Over-determined Laplacian Partial Differential Equations (ODETLAP) to “fill in” the missing data points. Our results show that the flow and watersheds on the reconstructed terrain are typically better than original ODETLAP point selection and lossy JPEG2000 compression.

1 Introduction

Terrain data is being sampled at ever increasing resolutions over larger geographic areas requiring special compression techniques to manipulate the data. Typically the effectiveness of a terrain compression technique is how well it minimizes the root mean square or the maximum error between the original terrain and the reconstructed geometry \cite{2}. This metric is not always the best choice for preserving hydrological information, since channels and ridges, essential for the calculation of drainage networks \cite{4}, might be lost. For example, selecting two points on opposite sides of a river can flatten the terrain and block water flow. Even without applying a lossy compression technique, drainage information is almost always lost in the data collection process. This causes a problem when trying to compute water flow, in particular when there are numerous small depression in the terrain that are inaccurately modeled as capturing water. The key to our implementation is to compute the drainage network on the inverted terrain, which captures the significant ridges. Omitting insignificant ridges can prevent small upward variations in the DEM (digital elevation model) from impeding water passage.

2 Prior Art

For calculating the drainage network for both the original terrain and the inverted terrain, a D8 model is used, where water at each cell in the terrain can flow in one of a possible eight directions. A flow accumulation grid is computed, where each cell contains an integer corresponding to how many other cells contribute flow to that point. Different from other methods that use flooding \cite{1}, our method computes flow using a system of linear equations \(Ax = b\) where \(x\) is a unknown \(N \times N\) length vector equal to the amount of water accumulation at each cell. Matrix \(A\) contains which cells receive water from adjacent neighbors and \(b\) is the initial flow or “rain” at each cell, usually equal to 1. Watersheds are computed using a very fast connected components program developed by Franklin \cite{3}. We deal with plateaus by determining the spill points and performing a breadth-first search.

Cells above a predefined threshold are considered significant and are added to the drainage and ridge network, which we call the ridge-river network. It is not necessary to store all these cells since they are clustered together and therefore add little value to a point selection compression technique. The Douglas-Peucker algorithm is used to reduce the number of points needed to represent each river segment to within a certain predefined error. The refined points can be stored and further compressed. To reconstruct the terrain we use an implementation of Over-determined Laplacian Partial Differential Equations (ODETLAP) \cite{2}. Each point is considered to be the average of its four neighbors, with the ridge-river points being known.

3 Results

Our reconstructed terrain captures the important aspects of the drainage network while still achieving a high compression rate. The reconstruction also has a more natural and realistic representation of the original hydrology because small insignificant ridges have been removed in the point selection process. This results in larger, fewer
watersheds. The recomputed drainage network is also captured accurately, besides the small tributaries which aren’t considered of high importance. We can also store the compressed version using far fewer points then the original DEM. The user can define the level of detail and hence the number of points by adjusting the tolerance level for the Douglas-Peucker algorithm.

Figure 1 shows the drainage network computed on four instances of a 400x400 elevation matrix representing a segment of the Hawaiian island of Oahu. In (a) the hydrology was computed on the original elevation matrix. (b) and (c) correspond to hydrology computed on the reconstructed terrain using ODETLAP, where in (b) the points were selected using our ridge-river technique described above, and in (c) using the original ODETLAP method (in each iteration the k “farthest points” were included). In (d) the hydrology is computed on a terrain is recovered from a lossy JPEG2000 compression. All the reconstructions have a similar RMS error of about 8.5.

4 Conclusions and Future Work

The first results showed that the hydrology consistency is better preserved on terrain recovered based on the ridge-river point selection method than using original ODETLAP point selection and JPEG2000. To confirm this observation, we intend to define a metric function to evaluate the hydrology preservation, that is, to verify how the river network and watershed are different on the original and recovered terrains. Another interesting investigation would be to use the drainage network as a model of natural terrain formation. This could be used to extract structure from the terrain for segmentation and division, allowing for better compression.

References


Drainage Network and Watershed Reconstruction on Simplified Terrain

PROBLEM

1. Large Dataset Size: Terrain data is being sampled at ever increasing resolution over larger geographic areas requiring special compression techniques to manipulate the data.
2. Sampling Issues: Dataset inaccuracies due to insufficient resolution and uncertainty in data collection errors impede water flow modeling small and unlithekt watersheds.

METHOD

1. Compute initial flow. Where every cell flows to the lowest adjacent neighbor.
2. Very fast connected components program is used to detect platform and sink.
3. We use a breadth first search to assign directions to flat regions.
4. Flow is recomputed using new directions.
5. Connected components program is used to determine watersheds.
6. Steps 1-5 are repeated for the inverse of the Terrain. This provides the Ridge Network.
7. Douglas-Peucker is used to detect the most significant points from the Ridge-River networks. These are stored as our final reconstructed representation of the terrain.
8. To reconstruct the terrain we use Overdetetermined Laplacian Pairs (Differential Equations (ODEs)) to fit the measured data points.
9. A ridge is defined as a non-platform point. Resulting in lower ink output, watersheds show flow at points well off erosion sources.

RESULTS

2. Better compression.
3. Terrain reconstructed without small insignificant edges.
4. Reconstructed terrain flows some sampling and data error.

RIDGE-RIVER NETWORK

merge networks that are not divided by significant ridges
merge networks that are divided by significant ridges

 Compared to the original terrain using deep points as the source

MERGE NETWORKS
Approximating Terrain with Over-determined Laplacian PDEs

Zhongyi Xie1, Marcus A. Andrade2,3, W. Randolph Franklin2, Barbara Cutler1, Metin Inanc1, Daniel M. Tracy1 and Jonathan Muckell2

1 CS Dept, Rensselaer Polytechnic Institute, Troy, New York, 12180–3590, USA
2 ECSE Dept, Rensselaer Polytechnic Institute, Troy, New York, 12180–3590, USA
3 Universidade Federal de Viçosa, Viçosa, Brazil

ABSTRACT

We extend Laplacian PDE by adding a new equation to form an over-determined system so that we can control the relative importance of smoothness and accuracy in the reconstructed surface. Benefits of the method include the ability to process isolated, scattered elevation points and the fact that reconstructed surface could generate local maxima, which is not possible in the original Laplacian PDE by the maximum principle. We use certain geometric algorithm including Triangulate Irregular Network, Visibility test, Level Set Component that discovers important points which reflect the terrain structure and use our extended Laplacian PDE to approximate the terrain from these points. We present experiments and measurements using different metrics and our method gives convincing results.

1. INTRODUCTION

Nowadays, the size of digital terrain data has grown to an extent that makes it essential to use some special representation or compression technique to manipulate the data. However, the development of processing and handling of digital terrain data has not advanced in pace with the data inflation. Elevation datasets are still stored as an elevation matrix. Common algorithms for compressing these matrices were originally designed for problems not specifically related to GIS and tend to yield poor results. For example, gzip, which USGS DEM data are usually compressed with, was originally designed as a plain text compressor.1 In Table 1, we list the compressed size using gzip for an ‘unfair’ comparison (because gzip is lossless).

In this paper, we use Over-determined Laplacian Partial Differential Equations (ODETLAP) to approximate and lossily compress terrains. We construct an over-determined system using points selected by one of the four strategies: triangulation, visibility tests, level set components and random selection; then use an over-determined PDE to solve for a smooth approximation. After that, we refine the approximation with respect to the original terrain by adding into the important points set points with biggest elevation error or slope error, and then use ODETLAP to solve for a better approximation. These two steps are alternately applied until the error is satisfactorily small.

ODETLAP can process not only continuous contour lines but isolated points as well. The surface produced tends to be smooth while preserving high accuracy to the known points. Local maxima are also well preserved. Alternative methods generally sub-sample contours due to limited processing capacity, or ignore isolated points.

2. OVER-DETERMINED LAPLACIAN APPROXIMATION OVERVIEW

Since we are working on single value terrestrial elevation matrix, We have the Laplacian equation

\[ 4z_{ij} - z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} = 0 \]  

(1)

for every unknown non-border point. In terrain modeling this equation has the following limitations:

- The solution of Laplace’s equation never has a relative maximum or minimum in the interior of the solution domain, this is called the “maximum principle”;2 so local maxima are never generated.
- The generated surface may droop if a set of nested contours is interpolated.

To avoid these limitations, an over-determined version of the Laplacian equation is defined as follows: apply the equation (1) to every non-border point, both known and unknown, and a new equation is added for a set \( S \) of known points:

\[ z_{ij} = h_{ij} \]  

(2)

where \( h_{ij} \) stands for the known elevations of points in \( S \) and \( z_{ij} \) is the “computed” elevation for every point, like in equation (1). The system of linear equations is over-determined, i.e., the number of equations exceeds...
the number of unknown variables, so instead of solving it for an exact solution (which is now impossible), an approximated solution is obtained by setting up a smoothness parameter \( R \) that determines the relative importance of accuracy versus smoothness.

### 3. ALGORITHM OUTLINE

The ODETLAP algorithm’s outline is shown in figure 1. Starting with the original terrain elevation matrix there are two point selection phases: firstly, the initial point set \( S \) is built and a first approximation is computed using the equations (1) and (2). Given the reconstructed surface, a stopping condition based on an error measure is tested. If this condition is not satisfied, the second step is executed. In this step, \( k \geq 1 \) points from the original terrain are selected according to the error in the reconstructed surface and are inserted in the existing point set \( S \); this extended set is used by ODETLAP to compute a more refined approximation. As the algorithm proceeds, the total size of point set \( S \) increases and the total error converges.

### 4. RESULT AND ANALYSIS

We test our algorithm on various real world terrain data sets. Each data set is a \( 400 \times 400 \) elevation matrix and original binary size is 320KB. In Table 1 we have results showing that our new compression scheme gets compression ratio of over 100 and the mean absolute error is no more than 2% in all cases. We use TIN to find the initial 100 important points and select 10 points in each of the 90 iteration, so at the end, the number of points we need to save is 1000.

### 5. FUTURE WORK

The next step of research consists of a few extensions in two directions: one is higher accuracy. We will investigate other PDEs to see if they can reconstruct the terrain more accurately than the Laplacian PDE. Another direction is higher compression. Currently we use lossless compression in the final compression step. We will test the use of lossy schemes, which can reach higher compression ratio at the cost of accuracy. Since slope is also a very important feature of terrain, we will also consider ways to minimize slope error in our representation.

### REFERENCES

Approximating Terrain with Over-Determined Laplacian PDEs

Zhongyi Xie, Marcus A. Andrade, W. Randolph Franklin, Barbara Cutler, Metin Inanc, Daniel M. Tracy and Jonathan Muckell
Rensselaer Polytechnic Institute, Troy, NY

Abstract:
We extend Laplacian PDE by adding a new equation to form an over-determined system (ODETLAP) which can be used to approximate the whole terrain from a few isolated points. We compress the terrain by selecting a few points which could later be losslessly decompressed using ODETLAP. Points selection algorithms include TIN, Visibility test, Level Set Components and Regular Selection.

ODETLAP:
Two sets of equations make the system over-determined Laplacian Equation: every non-border point is the average of its neighbors:
\[4z_i = z_{i-1} + z_{i+1} + z_{i-j} + z_{i+j}\]
New equation: Some points are already known:
\[z_i = \hat{z}_i\]
Use a smoothness parameter R to interpolate the two. R reflects the relative importance of accuracy vs. smoothness.
Adds capabilities to the classical system
- Local maxima inference
- Inconsistent data conflation

Encoding ODETLAP’s output:
- Code (x, y) separately from z:
- Run-length encode the bitmap:
- Delta code (z), then use bzip2.
- Approach information theoretic within 20%

1. Compression Results (TIN + Greedy ODETLAP)

<table>
<thead>
<tr>
<th>Data</th>
<th>Size (bytes)</th>
<th>Compression ratio</th>
<th>RMS Elev. Error m</th>
<th>RMS Slope Error deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>min1</td>
<td>2048</td>
<td>107.1</td>
<td>8.49</td>
<td>2.91</td>
</tr>
<tr>
<td>min2</td>
<td>5558</td>
<td>60:1</td>
<td>9.93</td>
<td>5</td>
</tr>
<tr>
<td>max1</td>
<td>11238</td>
<td>194:1</td>
<td>8.21</td>
<td>1.94</td>
</tr>
<tr>
<td>max2</td>
<td>9744</td>
<td>35:1</td>
<td>9.48</td>
<td>8.34</td>
</tr>
<tr>
<td>min3</td>
<td>8671</td>
<td>35:1</td>
<td>9.55</td>
<td>8.36</td>
</tr>
</tbody>
</table>

2. TIN + Greedy Elevation Comparison: MinD Dataset
(Compressed Size: 7641 bytes)

3. Accuracy vs. Size

- RMS Error vs. Terrain Size
- Slope Error vs. Terrain Size

Original Terrain
+ Reconstructed Terrain
+ TIN + Greedy ODETLAP
Slope Accuracy and Path Planning on Compressed Terrain

W. Randolph Franklin\(^1\), Daniel M Tracy\(^1\), Marcus A Andrade\(^{1,2}\), Jonathan Muckell\(^1\), Metin Inanc\(^1\), Zhongyi Xie\(^1\), and Barbara M Cutler\(^1\)

\(^1\) Rensselaer Polytechnic Institute  
Troy, New York, 12180–3590, USA  
frankwr@rpi.edu – http://wrfranklin.org/, tracyd@rpi.edu, muckej@rpi.edu, inancm@rpi.edu, xiez@rpi.edu, cutler@cs.rpi.edu  
\(^2\) DPI - UF Viçosa – Brazil

Abstract

We report on variants of the ODETLAP lossy terrain compression method where the reconstructed terrain has accurate slope as well as elevation. Slope is important for applications such as mobility, visibility and hydrology. One variant involves selecting a regular grid of points instead of selecting the most important points, requiring more points but which take less space. Another variant adds a new type of equation to the overdetermined system to force the slope of the reconstructed surface to be close to the original surface’s slope. Tests on six datasets with elevation ranges from 505m to 1040m, compressed at ratios from 146:1 to 1046:1 relative to the original binary file size, showed RMS elevation errors of 10m and slope errors of 3 to 10 degrees. The reconstructed terrain also supports planning optimal paths that avoid observers’ viewsheds. Paths planned on the reconstructed terrain were only 5% to 20% more expensive than paths planned on the original terrain. Tradeoffs between compressed data size and output accuracy are possible. Therefore storing terrain data on portable devices or transmitting over slow links and then using it in applications is more feasible.  

Keywords: terrain compression, slope accuracy, path planning, ODETLAP.

1 Introduction

As ever larger quantities of higher resolution terrain data become available, such as using IFSAR and LIDAR, more efficient compression techniques become more important. This is especially true when it is desired to store the data on portable devices or to transmit the data over slow links. High-resolution data may also compress differently when it is qualitatively different
Compression may be either *lossless*, where the restored data is identical to the original data, or *lossy*, where an error is introduced. This choice is not unique to terrain; audio data is also usually compressed lossily. Lossy compression is appropriate when the increased efficiency (i.e., the decreased size of the resulting file) is worth it, or when the original data is imperfect. That is, if the original data has an RMS error of 5 meters (m), then a compression algorithm introducing an average error of 0.5m is overkill.

The desired application for the terrain data influences the appropriate metric for evaluating the compression. The easy metric is RMS elevation error, Franklin and Said (1996). However, some parts of the terrain may be more important than others. For example, sharp points in the profile along the skyline are what viewers recognize. This author has had the experience of looking at a mountain range on the horizon while simultaneously looking at a commercial rendition of that same scene, and being unable to correlate the real world with the computer model. The problem resides in the computer model’s lack of high spatial frequencies. This may be caused by using calculus tools such as Fourier or Taylor series that assume that the terrain is differentiable many times, and that high frequencies are less important than low frequencies. Both assumptions are false. Not only does nothing in the physics of terrain formation select for smoothness, but rather the reverse. Erosion causes undercutting and slumping leading to cliffs, that is, elevation discontinuities.

**Slope** is one terrain property that is important to represent accurately. The slope of terrain influences *mobility* (it is difficult to drive up a cliff), *accessibility by air* (aircraft cannot land on a slope), *hydrology* (steeper slopes erode more quickly) and *visibility* (changes in slope are recognizable, and observers sited on a break in the slope may be able to see more).

Slope is often ignored because the assumption is that it comes for free once the elevations are represented sufficiently accurately. However, differentiating any imprecise function amplifies the errors. Also, from math analysis we know that approximating a function $f(x)$ more accurately, i.e., $\limsup_{i \to \infty} \left| f_i(x) - f(x) \right| \to 0$, gives no guarantees about $\limsup_{i \to \infty} \left| f'_i(x) - f'(x) \right|$, which may increase without bound. Indeed, it was such paradoxes that motivated the formalization of calculus in the 19th century.

The compression methods introduced here are extensions of ODETLAP, Franklin et al. (2007), and summarized in Figure 1. Briefly, ODETLAP solves a sparse overdetermined system of linear equations for the elevations $z_{ij}$ in an array where a few points’ elevations $h_{ij}$ are known. Each known point has an equation

$$z_{ij} = h_{ij}$$

(1)

Every non-border point, known or not has an equation

$$4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1}$$

(2)
Border points form a messy special case of no deep theoretical interest, but with the following practical difficulties. Not including equations for border points may lead to the system being underdetermined. A careless choice of equation may bias the surface to being horizontal at the borders, without physical justification.

Since there are more equations than unknowns, the system is overdetermined; we solve for a best fit. The two classes of equations are weighted differently, depending on the relative importance of accuracy versus smoothness. Weighting Equation 1 more highly relatively to Equation 2 makes the resulting surface more accurate but less smooth. A small degree of inaccuracy enables a large degree of smoothness. Indeed, a design requirement for ODETLAP was that, when interpolating between contour lines, that the contour lines not be visible in the resulting surface. Also, broken contours and even isolated points may be processed. These desirable properties are not always shared by competing surface fitting techniques.

![Diagram of ODETLAP Process]

**Fig. 1. ODETLAP Process**

There is little prior art on compressing slopes, apart from some descriptions of fundamental limits. A resolution of 25m or lower cannot identify steep slopes correctly Kienzle (2004). A resolution of 30m with elevations in meters results in a precision of slope calculations no better than 1.9°, Hunter and Goodchild (1997).
## 2 Terrain Data Structures

The underlying terrain data structure for the research presented in the paper is a matrix or array of elevations. There are other possibilities. One alternative would be high-order spherical harmonics as used in geopotential modeling. However, they are not as applicable to terrain, if only because their complexity grows quadratically with their accuracy. Wavelets of various types are used somewhat, and may become more popular in the future. The major alternative to an array of elevations is a Triangulated Irregular Network (TIN). Franklin (1973) did the first implementation (under the direction of Douglas...
and Peucker) of the TIN in Geographic Information Science. In the next section we use an updated version of that program, Franklin (2001). In contrast to Isenburg et al. (2006), Franklin (2001) operates incrementally, in the spirit of the Douglas-Peucker line generalization algorithm, Douglas and Peucker (1973) (independently discovered by Freeman and Ramer, Ramer (1972)). In each iteration, it greedily inserts the point that is farthest from the current surface. It can process arrays of up to $10^4 \times 10^4$ points in core. The time to completely TIN a level-I DEM with $1201^2$ points (until the max error is under 0.5m) is under 30 CPU seconds on a laptop. Also in contrast to Isenburg, it imposes no restrictions on the size of the generated triangles. However, because it operates out of core, Isenburg can process much larger datasets.

One disadvantage of a TIN compared to an array is the increased complexity of storing the data compactly, since in a naive implementation, most of the storage will be devoted to the topology. Also, rendering the terrain without producing a triangular appearance can require either very many triangles or a smoothing operator. Finally, representing slope accurately, one topic of this paper, appears problematic with a TIN. On the other hand, unlike an array a TIN is not tied to a particular coordinate system and can better represent large regions of the earth.

3 ODETLAP TIN+Greedy

The first question is, how well does ODETLAP represent slopes? Slope is qualitatively somewhat different from elevation: its autocorrelation distance is smaller, but it requires fewer significant bits.

We used six $400 \times 400$ test datasets, three hilly and three mountainous, extracted from level-2 DEMs. $400 \times 400$ is a resolution that we can easily process using the default sparse linear equation solver in Matlab; larger resolutions are possible with other techniques, such as the Paige-Saunders method used by Childs (2003, 2007). ODETLAP TIN+Greedy, the basic version of ODETLAP, selects points with the following two stage process.

1. Use our incremental triangulated irregular network (TIN) program to select $P$, an initial set of important points.
2. Fit a surface $S$ to $P$.
3. If $S$ is sufficiently accurate then stop.
4. Otherwise, find the 10 to 30 points of the original $400 \times 400$ points that are farthest from $S$. When forming this batch of points to insert, we assume that very close points are redundant, and require points to be at least a couple of pixels apart. Increasing this forbidden zone beyond that confers no additional advantage. Points are inserted in batches because of the time to recompute the surface in step 2.
5. Insert the new points into $P$.
6. Go back to step 2.
The \((x,y)\) are compressed by forming a \(400 \times 400\) bitmap showing the points’ locations, then compressing it with a runlength code. The resulting size is not much worse that the information-theoretic limit. The \(z\) are compressed with various methods such as \textit{bzip2}.

ODETLAP’s running time depends greatly on the input elevation matrix’s sparsity. Basic ODETLAP on a \(400 \times 400\) terrain took about 4.6 minutes when 5.4% of the elevations were known, and about 7.5 minutes on a 2.2GHz processor when 7.5% of the elevations were known. Denser input matrices required over 15 minutes.

Table 1 summarizes the results. ODETLAP TIN+Greedy compressed these terrains by factors ranging from 30:1 to 100:1 compared to the original binary file, with RMS elevation errors less than 10\,m and a slope error ranging from 1.7° to 8.4°, depending on the terrains’ ruggedness. The next question is, what is the tradeoff of size versus accuracy? Figures 3 and 4 answer this.

![Compressed Size vs. Error](image)

**Fig. 3.** ODETLAP Tin+Greedy Size – Elevation Accuracy Tradeoff

A major advantage of ODETLAP TIN+Greedy is that it selects the points in order of importance, and so permits progressive transmission of the points. However there will be a size penalty since compressing points incrementally is less efficient than compressing them in one set. Indeed, the former method stores the order of the points, which the latter does not. Therefore, for \(N\) points, the penalty will be at least \(N \lg N\) bits (the information content of selecting one permutation from \(N!\) permutations), but will probably be more. A larger storage cost of this method compared to the following one is caused by these points’ positions being irregular.
4 ODETLAP-Regular Grid

With this alternative, instead of greedily selecting the $N$ most important points, we select points on a regular grid uniformly spaced, say 40 $\times$ 40, or every 10th point in $x$ and $y$. The first advantage is that the points’ locations $(x, y)$ do not need to be stored. Second, since the $z$ form a regular array, using any image processing compression technique becomes easy. However, since ODETLAP-REGULAR GRID does not adapt to changes in the spatial complexity of the terrain, it will require more points and it may miss small features. Is this tradeoff worth it?

Table 2 shows the results. For each dataset, the number of points was increased, keeping a square grid of points but selecting more points equally spaced in columns and rows, until the RMS elevation error was under 10 m. For the same number of points, the compressed size varied slightly because
Table 3. ODETLAP Regular grid with DCT Results

<table>
<thead>
<tr>
<th>Elevation range</th>
<th>Hill 1</th>
<th>Hill 2</th>
<th>Hill 3</th>
<th>Mtn 1</th>
<th>Mtn 2</th>
<th>Mtn 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original binary size</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
</tr>
<tr>
<td>Compressed size</td>
<td>306B</td>
<td>807B</td>
<td>172B</td>
<td>2194B</td>
<td>2027B</td>
<td>2013B</td>
</tr>
<tr>
<td>Compression ratio</td>
<td>1046:1</td>
<td>397:1</td>
<td>1860:1</td>
<td>146:1</td>
<td>158:1</td>
<td>159:1</td>
</tr>
<tr>
<td># pts selected</td>
<td>529</td>
<td>1600</td>
<td>225</td>
<td>4489</td>
<td>4489</td>
<td>4489</td>
</tr>
<tr>
<td>RMS elevation error</td>
<td>9.6m</td>
<td>10.0m</td>
<td>9.7m</td>
<td>9.7m</td>
<td>10.0m</td>
<td>9.9m</td>
</tr>
<tr>
<td>RMS slope error</td>
<td>4.3°</td>
<td>6.5°</td>
<td>3.0°</td>
<td>10.°</td>
<td>10.°</td>
<td>9.9°</td>
</tr>
</tbody>
</table>

Different sets of $z$ compress differently. After achieving an RMS elevation error smaller than 10, the $z$ coordinate of the selected points are compressed using bzip2. Comparing with the ODETLAP TIN+Greedy results, the compression ratio is about 2 times better. On the other hand, the RMS slope error is a little worse.

As an extension, we lossy compressed $z$ as follows. The selected $z$ values were rounded off while preserving an RMS error less than 10 and then transformed with a Discrete Cosine Transform (DCT). A DCT, widely used in image compression, is similar to a Fourier series, but uses a set of higher and higher frequency square waves instead of sines and cosines to approximate a function. The more square waves are used, the more accurate the approximation is, but the more space it takes, Wikipedia (2008).

Then the resulting sequence was compressed using bzip2. For a given elevation or slope error, this method compresses better. See Table 3.

5 Path Planning

The next test of our compression algorithm was for path planning on terrain, where the traveler is hiding from a set of observers who have been optimally positioned, Franklin and Vogt (2006); Franklin (2002). That is, if we use the compressed terrain to plan a path, how good is that path? We chose the following metric, designed to incorporate several factors affecting real paths.

$$C = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \cdot \left(1 + \max \left(0, \frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}} \right) \right) \cdot (1 + 100v) \quad (3)$$

The first term says that shorter paths are better. The second says that moving uphill is expensive. The third term says that being seen by an observer is very expensive ($v = 1$ if the traveler is in sight, 0 otherwise). Note that the uphill term means that this metric is not symmetric; the optimal path from $a$ to $b$ has a different cost, and is not simply the reverse of, the optimal path from $b$ to $a$. Therefore some other path planning algorithms will fail. Further, since a $400 \times 400$ dataset has $400^2$ points, graph traversal algorithms
employing an explicit cost matrix are infeasible. Finally, some search strategies that climb hills in parameter space (unrelated to climbing hills on the terrain) stop at local optima, which is undesirable. To address all these concerns, we created a modified A* search procedure, Tracy et al. (2007), and used it to plan paths between many pairs of sources and destinations on each dataset. Figure 5 shows many paths plotted on the \textit{mtn3} dataset. Each little white lighthouse represents an observer. The surrounding colored region is the observer’s viewshed. Gaps in the viewsheds are caused by ridges hiding the terrain behind them. The dark regions of the figure are invisible to all the observers. Figure 5 also shows choke points in the terrain, which are traversed by many paths. Those would be candidates for siting future observers.

How to evaluate the path computed on the compressed terrain is also important, and the obvious choices may be wrong. For example, the cost of the path computed on the compressed terrain is meaningless. Indeed, if the terrain were compressed to be flat, then paths computed on it would have no cost for moving uphill and so would be artificially cheap, which is wrong. Even comparing the distance between two paths is meaningless for evaluating them.
Table 4. Increased cost of paths computed on compressed terrain

<table>
<thead>
<tr>
<th>Compressed Data size</th>
<th>Compression ratio increase</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1 1763</td>
<td>182</td>
<td>5.5%</td>
</tr>
<tr>
<td>Hill2 1819</td>
<td>176</td>
<td>6.1%</td>
</tr>
<tr>
<td>Hill3 1607</td>
<td>199</td>
<td>4.4%</td>
</tr>
<tr>
<td>Mtn1 1925</td>
<td>166</td>
<td>19.2%</td>
</tr>
<tr>
<td>Mtn2 1884</td>
<td>170</td>
<td>18.2%</td>
</tr>
<tr>
<td>Mtn3 1946</td>
<td>164</td>
<td>17.0%</td>
</tr>
</tbody>
</table>

Fig. 6. Compressed path evaluation algorithm

Indeed, two paths may be legitimately quite different but have the same cost; we don’t care. Our metric recognizes that the purpose of computing a path on any terrain, compressed or original, is to use it in the real world. Therefore, we transfer the path back to the original terrain dataset, and evaluate it there, as shown in Figure 6.
Table 4 shows the path inefficiency when our six terrains are compressed by factors of at least 164:1. Paths computed on these very compressed terrains were suboptimal by only 6% to 19%.

6 ODETLAP+Slope

With ODETLAP TIN+GREEDY, we insert points with the greatest absolute elevation error. Since the goal is to represent slopes accurately, one obvious improvement would be to insert points with large slope errors. Another possibility would be to insert groups of close points since fixing the elevations of a set of close points should also fix the slope in that neighborhood. Both these ideas, and many other experiments not detailed here, had disappointing results. It was time to extend the ODETLAP equations themselves.

Three different representations of the terrain need to be distinguished in order to understand this section.

Original representation This is the original 400 × 400 matrix of elevation posts that we wish to compress.

Compressed representation This compact version is what would be transmitted or stored on portable devices.

Reconstructed terrain The compressed representation would be reconstituted into this new 400 × 400 matrix in order to be used.

For ODETLAP+SLOPE, we supplement the two existing types of equations, 1 and 2 with a new type of equation designed to force the slope in x and y to be more accurate.

\[
\begin{align*}
    z_{i+1,j} - z_{i-1,j} &= h_{i+1,j} - h_{i-1,j} \\
    z_{i,j+1} - z_{i,j-1} &= h_{i,j+1} - h_{i,j+1}
\end{align*}
\]

This sets the $\Delta z$ between the northern and southern neighbors equal to its known value, and sets the $\Delta z$ between the western and eastern neighbors equal to its known value. The elevation of the center point is not used. It was done this way because these two $\Delta z$s are the values used by the Zevenbergen-Thorne method, Zhou and Liu (2004), a common method for computing slopes, Zevenbergen and Thorne (1987). (The cross product of the two vectors becomes the normal to the surface.) Our system permits the indices to be chosen arbitrarily, to allow for pairs of nonadjacent points to be used; this is a topic of potential future research.

Since the system is overconstrained, the relative weights of the different types of equations can be set depending on the relative importance of slope accuracy, elevation accuracy, or smoothness. The idea for this addition is that the extra freedom of allowing elevations to drift somewhat, provided that the slopes remain accurate, may allow greater slope accuracy.
Either ODETLAP TIN+Greedy or ODETLAP Regular grid may serve as the basis for adding equations 4 and 5. In the former case, we iterate the process of greedily inserting the points whose reconstructed slopes are the worst. ODETLAP TIN+Greedy requires fewer points but ODETLAP Regular grid requires less space to store each of the points on the grid (though any extra irregular points off the grid will take the same space as in ODETLAP TIN+Greedy). As before, we add points in batches for efficiency, and use forbidden zones around the points to prevent close pairs of points to be added in the same iteration, although a point $P$ added in one iteration may be adjacent to a point added in an earlier iteration, if $P$'s error is sufficiently large.

![Fig. 7. Slope accuracy vs number of points for Mtn2](image)

Figure 7 shows how three variants of this idea perform on the Mtn2 dataset. They are selecting points in a regular grid, greedily selecting irregular points, and greedily selecting irregular points using an $11 \times 11$ forbidden zone. The $x$-axis is the number of points in the compressed representation (out of a total of 160000 points). The $y$-axis shows the average and maximum slope errors (the three max curves are the higher ones). The best method is greedily selecting irregular points using a forbidden zone.
7 Conclusions and Future Work

Representing terrain, including slope, with ODETLAP has great potential. We are now exploring some of its variations, and applying it to high resolution urban data. The major problem to be addressed is the computation space and time required. We are also extending our path planning algorithm for road construction. Here, we are allowed to modify the terrain with cuts and fills when planning the path. Another application of ODETLAP is terrain smoothing, which might be applied to any other terrain representation. Indeed that ODETLAP was created to smooth or interpolate between contours so that those contours would not be visible in the resulting surface.

One problem with all compression techniques is that they do not preserve Hydrology. Regions of the world where the terrain was formed by erosion caused by surface water flow have distinctive properties. There are almost no actual local minima (basins, depressions), because they become lakes. In the few depressions in the coterminous USA, such as the Great Salt Lake, Salton Sea, and Crater Lake, the water either evaporates or percolates away. However, there are many fictitious depressions caused by errors in measuring the terrain or by insufficiently fine sampling, Maidment et al. (1997). That is, the water may exit a depression via a canyon that is so narrow that it fits between two adjacent elevation posts, and so is missed. We are now studying how the hydrological properties of the terrain under compression. This is an instance of the general problem of compressing multiple layers of cartographic data where preserving the relationships between the layers after reconstruction is at least as important as preserving the individual layers’ accuracy.

The most general problem is to construct the terrain from a set of mathematical operators that force the resulting terrain to have the desired properties. For instance, suppose that we carved the terrain out of a block of earth with a shovel, with repeated applications of the following operation. Place the shovel touching the earth at a some point. Move the shovel along any trajectory ending at the edge of the earth, provided that the shovel always gets lower and lower. Then, repeat with another shovel path, etc. The terrain that is created will never have an interior local minimum. That is, it will be hydrologically “correct”. Can we reduce this idea to practice?

8 Acknowledgement

This research was supported by NSF grants CCR-0306502 and DMS-0327634, by DARPA/DSO under the Geo* program, and by CNPq – the Brazilian Council of Technological and Scientific Development.
References


Douglas DH and Peucker TK (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Canadian Cartographer 10(2), 112–122


Franklin WR (2002) Siting observers on terrain. in D Richardson and P van Oosterom, eds. Advances in Spatial Data Handling: 10th International Symposium on Spatial Data Handling. Springer-Verlag pp. 109–120


URL: http://www.ce.utexas.edu/prof/maidment/atlas/esri97/ESRI.htm

Ramer U (1972) An iterative procedure for the polygonal approximation of plane curves. Computer Graphics and Image Processing 1, 244–256


Summary

• We're in a data-rich environment.
• We must compress terrain.
• Lossy compression is preferred.
• What metric should we use?
• Today, we'll extend ODETLAP to compress terrain (matrices of elevation data) so that the decompressed terrain’s slope is comparable to the original terrain’s slope.
Various terrain representations

- Fourier series.
- Spherical harmonics.
- Fractals.
- Wavelets.
- Triangulated Irregular Network\(^2\)
- Matrix of elevation posts.
  - Compact.
  - Simple enough that accessing the data is trivial.
  - Scales up.
  - Regular spacing in spite of irregular information density is not a problem if the compression algorithm can adapt. (This point is not always appreciated)

\(^2\)(Franklin, 1973)

Why consider slope?

Slope is important for
- mobility
- erosion
- aircraft
- visibility
- recognition
Bad commercial slope representation

Commercial SW:

Photo:
Accurate elevations $\not\Rightarrow$ accurate slopes

- Ignoring errors, slope is simply $f'(x)$
- But $\lim_{i \to \infty} |(f_i(x) - f(x))| \to 0$, gives no guarantees about $\lim_{i \to \infty} |(f'_i(x) - f'(x))|$.
- Consider two approximations to $y(x) = 0$

- Elevation got better but slope got worse.

ODETLAP hard example

- input: contours with sharp corners
- output: smooth silhouette edges, inferred top

Re=3, maxerr=12%, mean=2.7%

Re=1, maxerr=5.5%, mean=0.8%
ODETLAP – Overdetermined Laplacian

Fundamental representation for this work

- Small set of posts ⇒ complete matrix of posts
- Overdetermined linear system:
  - \( z_{ij} = h_{ij} \) for known points,
  - \( 4z_q = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} \) for all nonborder points.
  - Emphasize accuracy or smoothness by weighting the two types of equations differently.
- Fills contours to a grid.
- Fill missing data holes.
- Handles
  - incomplete contours,
  - complete contours,
  - kidney-bean contours,
  - isolated points,
  - inconsistent data.
ODETLAP TIN+greedy point insertion

- Use incremental TIN to find initial set \( P \) of approx 1000 important points.
- Fit surface with ODETLAP.
- If it's good enough, then stop.
- Find approx 30 worst points, and insert into \( P \).
- Loop back to step 2.

*Why 30?* Efficiency; ODETLAP takes minutes per run.

*Forbidden zone concept:* In the same step, don’t insert very close points.

Coding the points

*Goal is min size not fewest points*

- *Coding* \( \{(x, y, z)\} \) to minimize size is as important as selecting the points.
- Various approaches were presented elsewhere.
- Using more points is good, if they can be coded better.
- E.g., regular grid of points.
- If progressive transmission is not desired, then, for irregular points, use compressed bitmap for \( \{(x, y)\} \) and \( bzip2 \) for \( z \).
Slope definition, accuracy

- Zevenbergen-Thorne \(((p_{i-1,j} - p_{i+1,j}) \times (p_{i,j-1} - p_{i,j+1}))_z\)
- \(p_{ij}\) not used

Limits of slope accuracy

- 1m elevation resolution
- 30m post spacing
- slope precision: \(\arctan\left(\frac{1}{30}\right) \approx 3\% \approx 2^\circ\)

Info content

- Slope’s autocorrelation distance is smaller than elevation’s
- However, slope has less relative precision.

Level-II sample datasets

400 \times 400 elevation matrices, elevation range

- Hill1 505m
- Hill2 745m
- Hill3 500m
- Mtn1 1040m
- Mtn2 953m
- Mtn3 788m
Test 1: Do nothing

- Is doing nothing sufficient?
- *I.e.*, In practice, if the elevation is accurate, then is the slope accurate?
- Soso.

ODETLAP TIN+Greedy Results

Select initial set of points with TIN, repeatedly insert points with worst error, stop when RMS elevation error ≤ 10m.

<table>
<thead>
<tr>
<th></th>
<th>Hill1</th>
<th>Hill2</th>
<th>Hill3</th>
<th>Mtn1</th>
<th>Mtn2</th>
<th>Mtn3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elev range</td>
<td>505m</td>
<td>745m</td>
<td>500m</td>
<td>1040m</td>
<td>953m</td>
<td>788m</td>
</tr>
<tr>
<td>Orig size</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
</tr>
<tr>
<td>Compress size</td>
<td>2984B</td>
<td>5358B</td>
<td>1739B</td>
<td>9744B</td>
<td>9670B</td>
<td>9895B</td>
</tr>
<tr>
<td>Compress ratio</td>
<td>107:1</td>
<td>60:1</td>
<td>184:1</td>
<td>33:1</td>
<td>33:1</td>
<td>32:1</td>
</tr>
<tr>
<td># pts selected</td>
<td>1040</td>
<td>2080</td>
<td>520</td>
<td>4160</td>
<td>4160</td>
<td>4160</td>
</tr>
<tr>
<td>RMS elev err</td>
<td>8.49m</td>
<td>9.93m</td>
<td>8.31m</td>
<td>9.48m</td>
<td>9.55m</td>
<td>9.68m</td>
</tr>
<tr>
<td>RMS slope err</td>
<td>2.81°</td>
<td>5°</td>
<td>1.65°</td>
<td>8.34°</td>
<td>8.36°</td>
<td>7.87°</td>
</tr>
</tbody>
</table>
Test 2: ODETLAP on regular grid

- Use a regular grid of points instead of inserting them greedily.
- More points needed, but
- Points compress better.

For given compressed file size
- Elevation error is smaller, but
- Slope error is larger.

This idea failed.

Test 3: Pin down the elevation at sets of close points

- When inserting a point into known set, also insert some adjacent points
- Thesis: that will force the slope to be accurate there.
- Not really.
- Analogy Lagrangian interpolation.

This idea also failed.
Test 4: Extend ODETLAP

- Explicitly incorporate slope
- New overdetermined linear system:
  - unknowns: $z_{ij}$
  - known:
    - some $h_{ij}$,
    - some $\Delta_x h_{ij} \triangleq h_{i-1,j} - h_{i+1,j}$,
    - some $\Delta_y h_{ij} \triangleq h_{i,j-1} - h_{i,j+1}$,
  - for all nonborder points:
    \[ 4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} \]
    - for known $h_{ij}$: $z_{ij} = h_{ij}$
    - for known $\Delta_x h_{ij}$ and $\Delta_y h_{ij}$:
      \[ z_{i-1,j} - z_{i+1,j} = \Delta_x h_{ij} \]
      \[ z_{i,j-1} - z_{i,j+1} = \Delta_y h_{ij} \]

Mtn2 experiments

Slope error vs number of points
Mtn2 experiments

Number of sufficiently accurate points (out of 160K points)

Mtn2 experiments

Slope error vs compressed file size

mtn2, ODETLAP with slope equations, lossy encoding of delta z
Path planning

- Fun to combine various projects we’ve worked on.
- *multiobserver siting + path planning + surface compression with ODETLAP*
- *unique feature of our path planning:* plans around complicated obstacles (viewsheds) while minimizing complex objective:

\[
C = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \cdot \left(1 + \max\left(0, \frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}}\right)\right) \cdot (1+100v)
\]

Many optimal paths
Conclusions & Future

- Terrain can be compressed to as to represent slope more accurately.
- ODETLAP represents terrain efficiently.
- Faster ODETLAP; now using RPI’s IBM Blue Gene/L (32K processors, #7 in June 2007 top500.org list).
- ODETLAP + hydrology.
- Long term goal: procedural terrain representation, where the math captures the structure.
Progressive Transmission of Lossily Compressed Terrain

Zhongyi Xie¹, Marcus A. Andrade¹,², W Randolph Franklin¹, Barbara Cutler¹, Metin Inanc¹, Jonathan Muckell¹, and Daniel M. Tracy¹

¹ Rensselaer Polytechnic Institute, Troy, NY, USA
² Universidade Federal de Viçosa, Viçosa, Brazil
{xiez,frankwr,inancm,muckej,tracyd}@rpi.edu
cutler@cs.rpi.edu,marcus@dpi.ufv.br

Abstract. The distribution and management of spatial data require strategies for handling large amount of terrain data that are now available. Especially, data like LIDAR and Digital Elevation Model (DEM) which have been used in a large group of diversified users. In this paper, we propose a progressive terrain data transmission scheme based on the Over-determined Laplacian PDE (ODETLAP) which can achieve a compromise of high compression ratio and accuracy. The ODETLAP can be thought of as a compressor of original terrain data and using run length encoding as well as linear prediction we can reach a higher compression ratio. In general, this technique is capable of reducing a hilly DEM dataset to 1% of its original binary size and a mountainous, to 3%. The accuracy loss in elevation and slope are also discussed.

Key words: DEMs, Terrain Compression, Progressive Transmission, Levels of Details, Lossy Encoding.

1 Introduction

In the past few years the geographical information science (GIS) community has seen an explosion in the data volume and a great improvement in the accuracy of data acquisition. In particular, a huge volume of data about terrains is available usually represented as a Digital Elevation Models (DEM) that consists of a regular grid of samples storing the height value of the terrain surface. High-resolution DEMs (sampled at 3m or less) is now widely available [1] which make possible more realistic rendering of terrain features. Also, we see a great growth in mobile GIS services as GPS units.

This huge amount of data requires some special techniques to manipulate and/or transmit it and a strategy frequently used is to transmit a large terrain (or image) progressively. That is, the data is transmitted in successive blocks starting with a coarse simplification which is progressively refined sending more information (for example, more points or pixels). Thus, the receiver can visualize or execute some operations in advance, before receiving the whole data.
In this paper we describe a new method to transmit a terrain progressively based on ODETLAP (Over-determined Laplacian Approximation) [2] which is a method to reconstruct the terrain from a subset of points. As shown in section 5, the main advantage of this method is that it allows the terrain reconstruction using a very small number of points which is very interesting for progressive transmission since the user can start the terrain visualization or processing too early. In section 3, we present the progressive transmission method and in section 4, we give a short description of ODETLAP.

2 Related works

In general, most of methods developed for raster data transmission are based on image compression techniques since they can provide good compression with no (or low) loss of information. Some simple methods randomly select subsets of pixels from the image being transmitted and incrementally complete the image by adding pixels [3, 4]. Other more sophisticated strategies [5–7] use some hierarchical structure, such as quadtree, to partition the image and select more pixels from those parts containing more details. Thus, the image quality can be incrementally improved by transmitting/including points in those image parts.

Other methods, for example [8–13] are based on advanced compression techniques such as wavelets decomposition, JPEG compression and JPEG2000. In general, these compression methods decompose the data in rectangular sub-blocks and each block is transformed independently. Furthermore, the image data is represented as a hierarchy of resolution features and its inverse at each level provides sub-sampled version of the original image. Thus, it is quite natural to apply this strategy for progressive transmission.

Generally, the raster data transmission over the internet is used for visualization purpose and so, the raster progressive transmission methods are very efficient since visual meaning can be extracted from images at very low resolution. However, in some applications, the objects need to be manipulated and/or processed to compute or to extract some additional information and, in this case, a vector representation [14, 15] is used.

3 Progressive Transmission based on ODETLAP

Conventional terrain/map related software that requires the availability of all data does not support any operation before the transmission process ends. Despite the development of networking technology and hardware, terrain data are still relatively large (a few Giga bytes) compared to the network speed (below 1 Mega bytes for most users). Consequently, conventional transmission methods’ response time would inevitably make them far more than being interactive. So, this problem can be circumvented using progressive transmission where the idea is to successively send the terrain data starting with a coarse simplification, and on the user end, visualize or start some other operation in advance. Although
progressive transmission may extend the total transmission time due to the extra time needed for multiple operations, it has advantages over the traditional method in the following ways: firstly, the client does not have to wait until all the data are received before any operation or viewing can be done, which increases the interactivity of the system; secondly, the client can determine whether the current surface is accurate enough before all the data are sent and received. This can save the transmission time and resource need and increase the flexibility and efficiency of the system.

The method proposed in this paper is based on ODETLAP, a method to reconstruct the terrain from a subset of points (see section 4), and the progressive transmission system consists of a server which first selects a subset of original points and sends those points to a client that reconstructs the surface using ODETLAP. After that, the reconstructed terrain is evaluated. If necessary, the client would request more points from the server, which could result in a more accurate approximation of the terrain. Figure 1 shows the whole process. The user end can then do more operations based on the reconstructed surface. While other existing progressive transmission methods transmits all the data from the server to the client, our progressive transmission system has the advantage that only a subset of points need to be sent, which saves a lot of networking resource as well as transmission time. As long as some data points reach to the client side, the whole terrain can be approximated by the ODETLAP, so the user can see the terrain early. When more points arrive at the client side, it can reconstruct the terrain more accurately from the extended points set, which gives to the user a more detailed view of the terrain.

![Progressive transmission flowchart](image)

Fig. 1. Progressive transmission flowchart
4 Over-determined Laplacian Approximation

This section presents a short description of the Over-Determined Laplacian Approximation (ODETLAP); for more details see [2].

4.1 Definition

The ODETLAP is an extension to Laplacian equation

\[ 4z_{xy} = z_{x-1,y} + z_{x+1,y} + z_{x,y-1} + z_{x,y+1} \]  

(1)

This equation states that for every non-border point identified by coordinate \((x, y)\) in the elevation matrix, the elevation \(z_{xy}\) is equal to the average of its neighbors. This equation has the limitation of being unable to represent local maxima in terrain modeling, which is the reason for the inclusion of the equation:

\[ z_{xy} = h_{xy} \]  

(2)

where \(h_{xy}\) is the actual elevation value.

Equations (1) and (2) form a basis for the over-determined linear system. The system’s input is a series of points \((x, y, z)\) indicating the elevation of certain locations \((x, y)\). For those locations with known elevation, we have both equations, and for the rest locations, only equation (1) is used. The relative importance of two sets of equations is determined by a parameter \(R\) during the approximation/interpolation process. Weighting equation (2) over equation (1) results in a more accurate surface which sacrifices smoothness, while weighting equation (1) over equation (2) gives us a smooth surface.

The idea of ODETLAP is: when we have incomplete information about the actual elevation matrix, we can use the known value and the constraint (average of its neighbors) to approximate/interpolate the elevation value for every unknown and known point. So ODETLAP can be considered as a solver whose input is a set of known points \((x, y, z)\) and an interpolation parameter \(R\) and outputs the DEM matrix of the complete terrain. The benefits of ODETLAP includes the ability of handling continuous as well as broken contour lines of elevations, processing kidney-bean-shaped contours without giving fictitious at regions inside and infer local maxima from a series of contours.

4.2 Algorithm

Since ODETLAP is capable of reconstructing the whole DEM matrix from a few sparse input points \((x, y, z)\) (typical input size range from 1% to 10% of the original points), we can use it as a decompressor. Figure 2 presents the flowchart of our algorithm. The DEM firstly undergoes a points selection which pick a subset of points \(S\) as input to ODETLAP solver. Together with the contour lines/border points and some other user supplied points, ODETLAP solver would reconstruct from \(S\) the whole DEM matrix of elevations. So, this gives
us a initial approximation of the elevation matrix. However, due to our pursuit of higher compression ratio, we normally pick a very small subset of points \(|S| = 1000\) and consequently the error in the initial approximation normally is above the predefined accuracy. Thus, after obtaining the initial approximation, unless it is accurate enough (which happens when the original elevation matrix is mostly flat) some refinement steps are executed. In each step, approximated surface is compared with the original DEM and points that are farthest from the actual ones are picked with care to form a new set \(S\). We assume one point is sufficient to “correct” the points in its neighborhood. That is, multiple points in the neighborhood are redundant and thus, points added in the same step are checked against each other to avoid pairs that are too close (for example, less than 5 pixels apart). The refinement steps end when overall RMS error falls below the required accuracy limit.

This process can be easily adapted for progressive transmission doing the points selection in the server end, based on ODETLAP, and sending these points to the client where the terrain is reconstructed (also using ODETLAP). Notice that the client needs to know the value of smoothness parameter \(R\) used in the server end.

### 4.3 Points Selection

The initial set of points are selected based on an incremental triangulation using the Franklin’s algorithm [16] which builds a triangulated irregular network (TIN) using a greedy insertion method to approximate a surface. The incremental TIN starts triangulating 3 points randomly selected and iteratively splits the existing triangles by finding points that are farthest away. This strategy is used to define an initial set with \(k\) points.
4.4 Compressing Points

To improve the transmission ratio, the points to be transmitted are compressed using the following strategies: the \((x, y, z)\) coordinates are split into \((x, y)\) pairs and \(z\) alone. The former are compressed using an adapted run length encoding method, described in section 4.5, and the \(z\) sequence is compressed using a linear prediction method and \texttt{bzip2} compressor program. This splitting process gives a good compression ratio because the \((x, y)\) values are restricted to the matrix elevation size while the \(z\) values are more “arbitrary”.

4.5 Run Length Encoding

The coordinates \((x, y)\) are different from \(z\) because they distribute evenly within the range \([1, N]\), where \(N\) is the DEM size, while \(z\) values distribute more closely. The run-length encoding is a simple lossless compression technique which, instead of storing the actual values in the sequence, stores the value and the count of sequence containing the same data value. Run is just a consecutive sequence that contains the same data value in each element. Since the \((x, y)\) values correspond to positions in a matrix, then we need to store only a binary bitmap showing whether one point is used in \(S\) or not. Thus, given a binary matrix of size \(N \times N\), the method is the following: for each run length \(L\), test if

1. \(L < 254\), then use one byte for it
2. \(254 \leq L < 510\), use \texttt{FFFE} as a marker and use a second byte for \(L - 254\)
3. \(510 \leq L < 766\), use \texttt{FFFF} as a marker and use a second byte for \(L - 510\)
4. \(L \geq 766\), then use \texttt{FFFFFFF} as a two byte marker and use next two bytes for \(L\).

In general, most runs are below 512, that means for most runs we need only 1 byte to store it. Here we assume all runs are shorter than 65535, which is a reasonable value for terrains of size \(400 \times 400\).

4.6 Linear Prediction

Unlike \((x, y)\) coordinates, \(z\) contains more redundancy due to the inherent redundancy in the original terrain. Normally, terrain data contains a high degree of correlation and that means we may predict the elevation value from its neighbors. The method of linear predication has been very successful in image processing [17]. However, due to the processing overhead, only recently have such predictors been widely used [18].

The sequence \(z\) that we are going to compress consists of the elevation values of the selected points by previous mentioned algorithm. Because points appear in the order they were selected, we need to order them by their corresponding \(x, y\) coordinates so that during reconstruction process, the correlation can be reestablished.

There are several different ways to do linear prediction, and within JPEG standard, there are seven modes of prediction [19]. However, most of them use
neighborhood information from two dimensions which in our case does not exist. As a result, we are only using the simplest mode which predicts the next entry in the sequence by the previous one. That is, given the original data sequence, the predicted sequence is computed by predicting each element by the previous one. Their difference is stored in a new correction sequence, which will be compressed by some data compression software like bzip2.

Table 1 presents the results to compress 6 different datasets with 1000 points each one selected from 6 DEMs representing 3 hilly and 3 mountainous terrains (figure 3 shows an image of these terrains). The table includes the total size, in bytes, required by the method (RLE + LP) described above and the size required by bzip2 to compress the triple (x, y, z). As you can see, the RLE+LP method requires about 60% of the space required by bzip2.

<table>
<thead>
<tr>
<th>Size (in bytes)</th>
<th>Hill1</th>
<th>Hill2</th>
<th>Hill3</th>
<th>Mtn1</th>
<th>Mtn2</th>
<th>Mtn3</th>
</tr>
</thead>
<tbody>
<tr>
<td>compr. xy (RLE)</td>
<td>1250</td>
<td>1243</td>
<td>1279</td>
<td>1228</td>
<td>1244</td>
<td>1241</td>
</tr>
<tr>
<td>compr. z (LP)</td>
<td>1304</td>
<td>1354</td>
<td>1209</td>
<td>1456</td>
<td>1424</td>
<td>1503</td>
</tr>
<tr>
<td>RLE+LP</td>
<td>2554</td>
<td>2597</td>
<td>2488</td>
<td>2684</td>
<td>2668</td>
<td>2744</td>
</tr>
<tr>
<td>Bzip2</td>
<td>4136</td>
<td>4234</td>
<td>4025</td>
<td>4328</td>
<td>4416</td>
<td>4355</td>
</tr>
</tbody>
</table>

Table 1. Compression of 1000 points: split into xy bitmap and z sequence, then use run-length encoding (RLE) on xy and linear prediction (LP) on z.

To give an idea of the ODETLAP’s performance, table 2 shows some information about the compressed representation of the 6 terrains shown in figure 3. For each dataset, a series of tests were run using different number of initial points, ranging from 400 to 4000. When refinement is needed, 10 percent of points are added until to get the elevation Root-Mean-Square error below 10.

The running time of ODETLAP is not real-time, mainly due to the amount of time needed to solve the partial differential equation. Generally speaking, on a laptop with 2GHz and 1GB of RAM, the reconstruction of those 400 by 400 terrains takes about 1 minute.

5 Tests and Results

We have tested our algorithm on a benchmarks of six real DEMs shown in figure 3. Each DEM is on a 400 × 400 grid, with spacing of 30 meters. First three (a),(b) and (c) are hilly datasets while last three are mountainous. Table 3 shows a summary of the results obtained to transmit progressively the 6 terrains in 5 steps: initially, 10% of points are sent to the client; next, for each data set, we selected as many points as necessary to reduce the RMS to 75%, 50% and 25% of its initial value; finally, in the last step, we transmitted points to achieve
<table>
<thead>
<tr>
<th></th>
<th>Hill1</th>
<th>Hill2</th>
<th>Hill3</th>
<th>Mtn1</th>
<th>Mtn2</th>
<th>Mtn3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elev range (m)</td>
<td>505</td>
<td>745</td>
<td>500</td>
<td>1040</td>
<td>953</td>
<td>788</td>
</tr>
<tr>
<td>Stand. Dev. (m)</td>
<td>78.9</td>
<td>134.4</td>
<td>59.3</td>
<td>146.0</td>
<td>152.4</td>
<td>160.7</td>
</tr>
<tr>
<td>Orig size (KB)</td>
<td>320</td>
<td>320</td>
<td>320</td>
<td>320</td>
<td>320</td>
<td>320</td>
</tr>
<tr>
<td>Compr. size (Bytes)</td>
<td>2984</td>
<td>5358</td>
<td>1739</td>
<td>9744</td>
<td>9670</td>
<td>9895</td>
</tr>
<tr>
<td>Compr. ratio</td>
<td>107:1</td>
<td>60:1</td>
<td>184:1</td>
<td>33:1</td>
<td>33:1</td>
<td>32:1</td>
</tr>
<tr>
<td># pts selected</td>
<td>1040</td>
<td>2080</td>
<td>520</td>
<td>4160</td>
<td>4160</td>
<td>4160</td>
</tr>
<tr>
<td>Elev. RMS error (m)</td>
<td>8.49</td>
<td>9.93</td>
<td>8.31</td>
<td>9.48</td>
<td>9.55</td>
<td>9.68</td>
</tr>
<tr>
<td>Elev. RMS error (%)</td>
<td>1.68</td>
<td>1.33</td>
<td>1.66</td>
<td>0.91</td>
<td>1.00</td>
<td>1.23</td>
</tr>
<tr>
<td>Slope RMS error (°)</td>
<td>2.81</td>
<td>5.00</td>
<td>1.65</td>
<td>8.34</td>
<td>8.36</td>
<td>7.87</td>
</tr>
</tbody>
</table>

**Table 2.** ODETLAP's compression performance

Figure 3. All six 16-bits original terrain: 400 by 400 resolution

6 Conclusion and Future Work

In this paper, we presented a progressive transmission method based on an overdetermined Laplacian PDE (ODETLAP) terrain representation model. Using ODETLAP method, it is possible to achieve a lossy compression whose ratio...
Table 3. The progressive transmission of 6 terrains in 5 steps: 10% of points, as many points as necessary to reduce the RMS error to 75%, 50% and 25% of the initial value and finally, to achieve a RMS error smaller than 10.

Fig. 4. Progressive example: Mountainous datasets (Mtn1): Compressed sizes are 1.3KB, 3.6KB and 7.3KB

is more than 30:1 while keeping a reasonable error. This is highly useful when compression ratio is emphasized over accuracy. Based on the lossy compression technique, the progressive transmission scheme improves the performance of terrain transmission. Since the data being transmitted is no longer the whole terrain but a small subset of points, it is possible to achieve greater transmission throughput.

Acknowledgments. This research was supported by NSF grants CCR-0306502 and DMS-0327634, by DARPA/DSO/GeoStar, and by CNPq - the Brazilian Council of Technological and Scientific Development. We thank Prof. Franklin T. Luk and Jared Stookey for valuable discussions on terrain representation.
References

Efficient viewshed computation on terrain in external memory

Marcus V. A. Andrade¹,², Salles V. G. Magalhães¹, Mirella A. Magalhães¹, W. Randolph Franklin², and Barbara M. Cutler³

¹DPI, Universidade Federal de Viçosa, Brasil
²ECSE Dept., Rensselaer Polytechnic Institute, USA
³CS Dept., Rensselaer Polytechnic Institute, USA

Abstract

The recent availability of detailed geographic data permits terrain applications to process large areas at high resolution. However the required massive data processing presents significant challenges, demanding algorithms optimized for both data movement and computation. One such application is viewshed computation, that is, to determine all the points visible from a given point $p$. In this paper, we present an efficient algorithm to compute viewsheds on terrain stored in external memory. In the usual case where the observer’s radius of interest is smaller than the terrain size, the algorithm complexity is $\theta(\text{scan}(n^2))$ where $n^2$ is the number of points in an $n \times n$ DEM and $\text{scan}(n^2)$ is the minimum number of I/O operations.
required to read $n^2$ contiguous items from external memory. This is much faster than existing published algorithms.

1 Introduction

Terrain modeling is an important application in Geographic Information Science (GIS). One aspect is the computation of all points that can be viewed from a given point (the observer). The region composed of the visible points is called the viewshed [15, 19]. This problem has many applications, such as determining the minimum number of cellular phone towers required to cover a region [5, 9, 11], optimizing the number and position of guards to cover a region [14, 20], analyzing influences on property prices in an urban environment [25], and optimizing path planning on a DEM [26].

The recent technological advances in data collection (such as LIDAR and IFSAR) have produced a huge volume of data about the Earth’s surface [31]. For example, modeling a $100 \text{ km} \times 100 \text{ km}$ terrain at $1\text{m}$ resolution requires $10^{10}$ points. Since most computers cannot store or process this huge volume of data internally, an external memory algorithm is required. Since the time required to access and transfer data to and from external memory is generally much larger than the internal processing time, the algorithm must try to minimize the external memory I/O [4, 21]. More specifically, such external memory algorithms should be optimized under a computational model whose cost is the number of data transfer operations instead of CPU time. One such model was proposed by Aggarwal and Vitter [1].

In this work, we present an efficient algorithm to compute the viewshed of a point on a terrain stored in external memory. The algorithm is an adaptation of Franklin and
Ray’s method [18, 19] to allow efficient manipulation of huge terrains (6GB or more). The large number of disk accesses is optimized using the STXXL library [13]. Our algorithm is more than six times faster and much easier to implement than the excellent algorithm proposed by Haverkort et al. [22].

This paper is organized as follows. Section 2 gives a brief description of viewshed computation and I/O-efficient algorithms for general problems as well as for viewshed computation. Section 3 formally presents the viewshed concepts. Section 4 briefly describes the I/O-efficient computational model. Section 5 describes the algorithm in detail. Section 6 analyzes its complexity. Section 7 gives the tests results, and Section 8 presents the conclusions.

2 Related Work

2.1 Terrain representation

Terrain is generally represented either by a Triangulated Irregular Network (TIN) or a Raster Digital Elevation Model (DEM) [16, 27]. A TIN, first implemented in GIS by Franklin [17], is a partition of the surface into planar triangles, i.e., a piecewise linear triangular spline. The elevation of any point $p$ is a bilinear interpolation of the elevations of the three vertices of the triangle whose projection onto the $xy$ plane contains the projection of $p$.

A DEM is simply a matrix of the elevations at regularly spaced positions or posts. The spacing may be either a constant number of meters or a constant angle in latitude and longitude.

Both representations are seen because neither is clearly better than the other [16, 24].
A DEM uses a simpler data structure, is easier to analyze and has higher accuracy at high resolution, but requires more memory space than a TIN. In this paper, we use the simpler, DEM data structure.

2.2 Terrain visibility

Terrain visibility has been widely studied. Stewart [30] shows how the viewshed can be efficiently computed for each point of a DEM. His interest is positioning radio transmission towers. Kreveld [32] proposes a sweep-line approach to compute the viewshed in \( \theta(n \log n) \) time\(^1\) on a \( \sqrt{n} \times \sqrt{n} \) grid. In [18, 19], Franklin and Ray describe experimental studies for fast implementation of visibility computation and present several programs that explore trade-offs between speed and accuracy. Young-Hoon, Rana and Wise in [33] analyze two strategies to use the viewshed for optimization problems. Ben-Moshe et al. [6, 7, 8] have worked on visibility for terrain simplification and for facility positioning. For surveys on visibility algorithms, see [15, 28].

2.3 External memory processing

Aggarwal and Vitter [1] discuss some problems, and propose a computational model to evaluate algorithm complexity based on the number of input/output (I/O) operations. Goodrich et al. [21] present some variants for the sweep plane paradigm considering external processing, while Arge et al. [4] describe a solution for externally processing line segments in GIS. This technique was also used to solve hydrology problems, such as computing water flow and watershed [3] on huge terrain.

\(^1\)\( \theta(f(n)) \) grows proportionally to \( f(n) \) as \( n \to \infty \). Formally, \( g(n) = \theta(f(n)) \Rightarrow \exists n_0 > 0, c_1 > c_2 > 0 \) such that \( n > n_0 \Rightarrow c_1 f(n) > g(n) > c_2 f(n) \). Hein [23, page 334].
Recently, Haverkort et al. [22] adapted van Kreveld’s method to compute viewsheds on terrain stored in external memory; its I/O complexity is $\theta(sort(n))$, where $n$ is the number of points on the terrain and $\theta(sort(n))$ is the minimum number of I/O operations required to sort $n$ contiguous items stored in external memory.

EMVS, which we present here, also has a worst-case I/O complexity of $\theta(sort(n))$, but its execution time is lower because of a more efficient strategy. Also, it is much easier to implement than the Haverkorth et al. method.

3 The Viewshed Problem

Visibility problems can be classified into two major categories: visibility queries and the visibility structure computation. Visibility queries consist of checking whether a given point $P$, the target, is visible from another point $O$, the observer or source. Both $O$ and $P$ are usually slightly above the terrain. For example, consider a radio tower communicating with a cellphone user. $P$ is visible from $O$ iff a straight line, the line of sight, from $O$ to $P$ is always strictly above the terrain. See Figure 1.

Figure 1: Point Visibility: $p_1$ and $p_4$ are visible from $p_0$; $p_2$ and $p_3$ are not visible from $p_0$. 
The visibility structure computation consists of determining some features such as the horizon and viewshed. Formally, the viewshed of a point $p$ on a terrain $T$ can be defined as:

$$\text{viewshed}(p) = \{ q \in T \mid q \text{ is visible from } p \}$$

When computing the viewshed, it is common to consider only points within a given distance $r$, the radius of interest, of $p$. In this case,

$$\text{viewshed}(p, r) = \{ q \in T \mid \text{distance}(p, q) \leq r \text{ and } q \text{ is visible from } p \}$$

We will generally assume $r$ and simplify the notation of $\text{viewshed}(p, r)$ to $\text{viewshed}(p)$. Since we are working with raster DEMs, we represent a viewshed by a square $2r \times 2r$ matrix of bits.

4 I/O efficient Algorithms

As mentioned before, for large datasets, I/O is the bottleneck. However, many GIS algorithms are designed to optimize internal processing, and so do not scale up. A more appropriate computational model is needed. One common model, proposed by Aggarwal and Vitter [1], defines an I/O operation as the transfer of one disk block of size $B$ between external and internal memory. The measure of performance is the number of such I/O operations. The internal computation time is assumed to be comparatively insignificant.

An algorithm’s complexity is related to the number of I/O operations performed by fundamental operations such as scanning or sorting $N$ contiguous elements stored in
external memory. Those are

\[
\begin{align*}
\text{scan}(N) &= \Theta \left( \frac{N}{B} \right) \\
\text{sort}(N) &= \Theta \left( \frac{N}{B \log \left( \frac{M}{B} \right)} \right)
\end{align*}
\]

where \( M \) is the internal memory size. Because \( B \gg 1 \), usually \( \text{scan}(N) < \text{sort}(N) \ll N \), and it is important to organize the data in external memory to decrease the number of I/O operations.

## 5 External Memory Viewshed Computation (EMVS)

Our algorithm, External Memory Viewshed (EMVS), is based on the method proposed by Franklin and Ray [19] that computes the viewshed of a point on a terrain represented as an internal memory matrix. That is summarized below.

### 5.1 Franklin and Ray’s Method

Given a terrain \( T \) represented by an \( n \times n \) elevation matrix \( M \), a point \( p \) on \( T \), a radius of interest \( r \), and a height \( h \) above the local terrain for the observer and target, this algorithm computes the viewshed of \( p \) within a distance \( r \) of \( p \), as follows:

1. Let \( p \)'s coordinates be \((x_p, y_p, z_p)\). Then the observer \( \mathcal{O} \) will be at \((x_p, y_p, z_p + h)\).
2. Imagine a square in the plane \( z = 0 \) of side \( 2r \times 2r \) centered on \((x_p, y_p, 0)\).
3. Iterate through the cells \( c \) of the square’s perimeter. Each \( c \) has coordinates \((x_c, y_c, 0)\), where the corresponding point on the terrain is \((x_c, y_c, z_c)\).

7
(a) For each \( c \), run a straight line in \( \mathcal{M} \) from \((x_p, y_p, 0)\) to \((x_c, y_c, 0)\).

(b) Find the points on that line, perhaps using Bresenham’s algorithm. In order from \( p \) to \( c \), let them be \( q_1 = p, q_2, \ldots, q_{k-1}, q_k = c \). A potential target \( D_i \) at \( q_i \) will have coordinates \((x_i, y_i, z_i + h)\).

(c) Let \( m_i \) be the slope of the line from \( O \) to \( D_i \), that is,

\[
m_i = \frac{z_k - z_i + p}{\sqrt{(x_i - x_p)^2 + (y_i - y_p)^2}}.
\]

(d) Let \( \mu \) be the greatest slope seen so far along this line. Initialize \( \mu = -\infty \).

(e) Iterate along the line from \( p \) to \( c \).

i. For each point \( q_i \), compute \( m_i \).

ii. If \( m_i < \mu \), then mark \( q_i \) as hidden from \( O \), that is, as not in the viewshed (which is simply a \( 2r \times 2r \) bitmap).

iii. Otherwise, mark \( q_i \) as being in the viewshed, and update \( \mu = m_i \).

The total execution time is linear in \( N \), the number of points in the terrain. In contrast, earlier algorithms that ran a separate line of sight to each potential target had an execution time of \( \theta(N^{3/2}) \).

This algorithm can be used to compute the viewshed on terrain in external memory. However, since the cells are accessed in a sequence defined by the radial sweep, that would require random access to the file, and the execution time would be unacceptably long. This random access order can be avoided using the adaptation described below.
5.2 The EMVS algorithm

The basic idea is to generate a list with the terrain points sorted by the processing order, that is, the points will appear in the list in the sequence given by the radial sweep and by the processing order along each line of sight. Thus, during the viewshed computation, the algorithm follows the list, avoiding accessing the file randomly.

This list stored in external memory is managed by STXXL (the Standard Template Library for Extra Large Data Sets) [13], which implements containers and algorithms to process huge volumes of data. This library allows an efficient manipulation of external data and, as stated by the authors, “it can save more than half the number of I/Os performed by many applications”.

Specifically, the algorithm creates a list $L$ of pairs $(c, i)$ where $c$ is a terrain position and $i$ is an index indicating the order in which $c$ should be processed.

To compute the cell indices, the lines of sight (originating at the observer $p$) are numbered in counterclockwise order starting at the positive $x$-axis, which is number 0 — see Figure 2. Thus, the cells are numbered increasingly along each line of sight; when a line of sight ends, the enumeration proceeds from the observer (again numbered) following the next line of sight. Of course, a same cell (point) can receive multiple indices since it can be intercepted by many lines of sight, especially if it is close to the observer. This means that a same point can appear in multiple pairs in the list $L$, but each pair will have a different index. Also, if the observer is near to the terrain border, that is, if the distance between the observer and the terrain border is smaller than the radius of interest $r$, some cells in a line of sight can be outside the terrain. In this case, those cells still will be numbered but they will be ignored (i.e. they will not be inserted in the list $L$). This is done to simplify the indices computation by avoiding many additional conditional tests.
Even when computing the cell indices as described above, the file still would be randomly accessed, as in the original algorithm. So, to build the list $L$, the algorithm reads the terrain cells sequentially from the external file and for each cell $c$, it determines (the number of) all lines of sight that intercept that cell.

Since the cells have a finite size (they are not points), we can determine the cells intercepted by a line of sight using a process similar to line rasterization [10]. That is, let $s$ be the side of each (square) cell and suppose the cell is referenced by its center. Also, let $a$ be a line of sight whose slope is $\alpha : 0 < \alpha \leq 45^\circ$. So, given a cell $c = (c_x, c_y)$, see Figure 3, the line of sight $a$ “intersects” the cell $c$ if and only if the intersection point between $a$ and the vertical line $c_x$ is a point in the segment $(c_x, c_y - 0.5s)$ and $(c_x, c_y + 0.5s)$; more precisely, given $(q_x, q_y) = a \cap c_x$, $a$ intersects $c$ if and only if $c_y - 0.5s \leq q_y < c_y + 0.5s$. In this case, a line of sight as the dashed line in the Figure 3 is assumed to intersect the cell above $c$.

$^2$For $45^\circ < \alpha \leq 90^\circ$, interchange $x$ and $y$ and use a similar idea.
Then, all lines of sight intersecting the cell $c$ are those between the two lines connecting the observer to the points $(c_x, c_y - 0.5s)$ and $(c_x, c_y + 0.5s)$ — Figure 3. Let $k_1$ and $k_2$ be the numbers of these two lines respectively. Considering the line of sight definition (a segment connecting the observer and the center of cells on the square border) and the line enumeration, we have that the number of the lines intersecting the cell $c$ can be determined by the cells in the border whose center are between $k_1$ and $k_2$ - see Figure 4. For example, in this Figure, cell $c$ is intersected by the lines 3 and 4.

Now, given a cell $c$, let $\kappa$ be the number of a line of sight intercepting $c$. Then, the index $i$ of the cell $c$ associated with $\kappa$ is given by the formula $i = \kappa \ast n + d$, where $n$ is the number of cells in each ray (this number is constant for all rays) and $d$ is the (horizontal or vertical) distance between points $c$ and $p$ — see Figure 5. Note that the distance $d$ is defined as the maximum between the number of rows and columns from $p$ to $c$.

Next, the list $L$ is sorted by the elements’ index, and then the cells are processed in the sequence given by the sorted list. When a cell $c$ is processed, all the “previous” cells...
that could block the visibility of $c$ have already been processed. So, the visibility of $c$
can be computed, as described before, just by checking the height of the cells along the
line of sight. When a cell located on the square border is processed, it means that the
line of sight processing has finished. The next cell in the list will be the observer’s cell,
indicating that the processing of a new line of sight will start.

To improve the algorithm’s efficiency, another list $L'$ (also stored externally and man-
aged by $STXXL$) is used to keep only the visible cells. When the algorithm determines
that a cell $c$ is visible, this cell is inserted in $L'$. The size of $L'$ is much smaller than
\(L\) since \(L'\) does not keep the indices, and usually many points are not visible. To avoid random access, before storing the viewshed, the list \(L'\) is sorted lexicographically by \(x\) and \(y\). Then the viewshed is stored in an external file where a visible position (a point in \(L'\)) is indicated by 1 and not visible point by 0.

Additionally, a piece of the terrain matrix is stored in the internal memory. The idea is to store the cells around the observer since these cells are processed more times than the farther ones. All cells inside a square centered at the observer position are stored internally, and are not inserted in the lists \(L\) and \(L'\). When a cell needs to be processed, the algorithm checks if it is in the internal memory. If so, the cell is processed normally; otherwise, it is read from list \(L\).

### 6 Algorithm complexity

Let \(T\) be some terrain represented by an \(n \times n\) elevation matrix. Let \(p\) be the observer’s position, and \(r\) be the radius of interest. As described in Section 5.2, the algorithm considers the cells that are inside the \(2r \times 2r\) square centered at \(p\). Assuming that each cell’s side is \(s\), there are, at most\(^3\), \(8r/s\) cells on the square’s perimeter (each square side has \(2r/s\) cells). Let \(K = r/s\). Thus, the algorithm shoots \(8K\) lines of sight and since each line of sight has \(K\) cells, the list \(L\) has, in the worst case, \(\theta(K^2)\) elements.

In the first step, the algorithm does \(\frac{n^2}{B}\) I/O operations to read the cells and build the list \(L\). Next, the list with \(\theta(K^2)\) elements is sorted and then it is swept to compute the

\[^3\]If the observer is close to the terrain border, the square might not be completely contained in the terrain.
cell’s visibility. Thus, the total number of I/O operations is:

\[
\theta \left( \frac{n^2}{B} \right) + \theta \left( \frac{K^2}{B} \log(\frac{K}{B}) \right) + \theta \left( \frac{K^2}{B} \right)
\]

Since, in general, \( r \ll n \), then \( K < n \) which implies that the number of I/O operations is \( \theta(\frac{n^2}{B}) = \theta(\text{scan}(n^2)) \). In the rare worst case when \( r \) is big enough to cover almost the whole terrain, the number of I/O operations is

\[
\theta \left( \frac{K^2}{B} \log(\frac{M}{B}) \left( \frac{K^2}{B} \right) \right) = \theta(\text{sort}(K^2))
\]

The algorithm also uses an additional external list \( L' \) to keep the visible cells and this list needs to be sorted. Since the list size is much smaller than the size of \( L \), the number of I/O operations executed in this step does not change the algorithm complexity.

7 Experimental results

EMVS was implemented in C++, in g++ 4.1.1 under Mandriva Linux, on a 2.8 GHZ Pentium PC with 1 GB of RAM and an 80 GB 7200 RPM serial ATA hard drive. To better evaluate the I/O operation cost, we considered two configurations. The first used the whole 1 GB of RAM and allowed our program to use 800 MB for data. The second used only 256 MB and allowed 200 MB for data. Although 256 MB main memory is quite small for modern PCs, we used it for two reasons: to illustrate the behaviour and trends of the algorithm as the difference between dataset and memory size increases, and to give an idea of the algorithm performance on portable devices with big hard drives but little
The tests used the data sets (from NASA STRM home page [29]) of the regions 1, 2 and 3 shown in Figure 6 sampled with a resolution of 1 arc of second (approximately 30m). From this terrain, we selected pieces of different sizes and each piece was obtained by defining the observer position and the radius of interest\(^4\). Since these datasets contain a very small percentage (less than 1.5%) of no-data (i.e., points for which the elevation is unknown or invalid), our results are not influenced by no-data points.

![Figure 6: Regions 1, 2 and 3 of the USA used in the tests.](image)

Tables 1 and 2 show the **EMVS** execution time with either 1G or 256 MB of RAM. We always considered the worst case radius of interest, i.e., big enough to cover the whole terrain. External processing time (I/O, external sorting, file access, etc) is shown separately (column Ext.) from the total running time (column Tot). To evaluate the

\(^4\)To compare the efficiency of our algorithm and the Haverkort et al. algorithm, we used terrain of size similar to those used by them.
influence of the number of visible points (the column \# Vis. Pts.) on the time, the observer was positioned at different heights (1, 50, 100, 1000 and 10000 meters) above the terrain. 1000 and 10000 meters are presented to demonstrate the algorithm’s scalability.

### Running time with 1GB RAM

<table>
<thead>
<tr>
<th>Terrain Size</th>
<th>Observing Height</th>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Vis. Pts.</td>
<td>Time (sec.)</td>
<td># Vis. Pts.</td>
<td>Time (sec.)</td>
</tr>
<tr>
<td>1</td>
<td>1.2 \times 10^6</td>
<td>17 43 4.5 \times 10^4</td>
<td>19 48 2.1 \times 10^4</td>
<td>20 47 6.3 \times 10^7</td>
</tr>
<tr>
<td>9029^2</td>
<td>50 6.7 \times 10^6</td>
<td>17 45 3.6 \times 10^7</td>
<td>17 50 6.3 \times 10^7</td>
<td>18 57 6.4 \times 10^7</td>
</tr>
<tr>
<td>311 MB</td>
<td>10^2 1.1 \times 10^7</td>
<td>17 46 6.3 \times 10^7</td>
<td>18 56 6.4 \times 10^7</td>
<td>18 57 6.4 \times 10^7</td>
</tr>
<tr>
<td></td>
<td>10^3 5.6 \times 10^7</td>
<td>18 54 6.3 \times 10^7</td>
<td>19 59 6.4 \times 10^7</td>
<td>20 59</td>
</tr>
<tr>
<td></td>
<td>10^4 6.3 \times 10^7</td>
<td>19 56 6.3 \times 10^7</td>
<td>19 59 6.4 \times 10^7</td>
<td>20 59</td>
</tr>
<tr>
<td>17150^2</td>
<td>50 4.2 \times 10^5</td>
<td>67 209 1.3 \times 10^5</td>
<td>70 208 1.1 \times 10^5</td>
<td>69 209 1.6 \times 10^5</td>
</tr>
<tr>
<td>1122 MB</td>
<td>10^2 9.5 \times 10^6</td>
<td>68 210 2.1 \times 10^8</td>
<td>66 243 2.1 \times 10^8</td>
<td>67 247 2.1 \times 10^8</td>
</tr>
<tr>
<td></td>
<td>10^3 1.5 \times 10^8</td>
<td>67 237 2.2 \times 10^8</td>
<td>66 246 2.2 \times 10^8</td>
<td>67 238</td>
</tr>
<tr>
<td></td>
<td>10^4 2.2 \times 10^8</td>
<td>69 249 2.2 \times 10^8</td>
<td>67 244 2.2 \times 10^8</td>
<td>67 248</td>
</tr>
<tr>
<td>33433^2</td>
<td>50 1.2 \times 10^9</td>
<td>1716 2598 4.3 \times 10^4</td>
<td>1766 2652 1.1 \times 10^9</td>
<td>1756 2630</td>
</tr>
<tr>
<td>4264 MB</td>
<td>10^2 7.3 \times 10^8</td>
<td>1759 2624 1.1 \times 10^7</td>
<td>1756 2654 2.2 \times 10^6</td>
<td>1763 2709 6.6 \times 10^6</td>
</tr>
<tr>
<td></td>
<td>10^3 1.5 \times 10^8</td>
<td>1769 2648 1.9 \times 10^8</td>
<td>1776 2728 2.2 \times 10^8</td>
<td>1763 2709 6.6 \times 10^6</td>
</tr>
<tr>
<td></td>
<td>10^4 6.4 \times 10^8</td>
<td>1716 2716 8.5 \times 10^8</td>
<td>1764 2821 8.6 \times 10^8</td>
<td>1776 2851 8.6 \times 10^9</td>
</tr>
<tr>
<td>40000^3</td>
<td>50 3.5 \times 10^6</td>
<td>3117 4546 3.9 \times 10^6</td>
<td>3981 5434 2.1 \times 10^8</td>
<td>3109 4599</td>
</tr>
<tr>
<td>6103 MB</td>
<td>10^2 2.9 \times 10^8</td>
<td>3108 4555 7.6 \times 10^7</td>
<td>3960 5530 1.7 \times 10^9</td>
<td>4290 5906 6.4 \times 10^8</td>
</tr>
<tr>
<td></td>
<td>10^3 7.7 \times 10^8</td>
<td>4241 5697 1.9 \times 10^8</td>
<td>4401 6046 6.4 \times 10^8</td>
<td>4603 6739 1.0 \times 10^9</td>
</tr>
<tr>
<td></td>
<td>10^4 1.2 \times 10^8</td>
<td>5064 6962 1.2 \times 10^9</td>
<td>4852 6808 1.0 \times 10^9</td>
<td>4544 6431</td>
</tr>
</tbody>
</table>

Table 1: **EMVS** running time (in seconds) at 1 GB RAM on pieces of terrain with different sizes from Regions 1, 2 and 3 and varying the observer height above the terrain (generating viewshed with different number of visible points - shown in the column \# Vis. Pts). In all cases, the radius of interest cover the whole terrain.

Figure 7 summarizes the internal and external processing time on Region 1 terrain,
Table 2: **EMVS** running time (in seconds) at 256 MB RAM on pieces of terrain with different sizes from Regions 1, 2 and 3 and varying the observer height above the terrain (generating viewshed with different number of visible points - shown in the column # Vis. Pts). In all cases, the radius of interest cover the whole terrain.

using 256 MB and 1 GB of RAM (the results for Regions 2 and 3 are quite similar). As expected, the external processing time is much larger than the internal processing time on terrain that is much bigger than the internal memory size. See charts (c), (d) and mainly (b) where the external processing time is longer (resp. shorter) than the internal
processing time when using 256 MB (resp. 1 GB). The difference in (b) occurs because the 1122 MB terrain can be processed almost completely in 1 GB internal memory, requiring few I/O operations. However, with only 256 MB, many I/O operations are necessary.

![Figure 7: The internal and external processing time using 256MB and 1GB of RAM on pieces of terrain from Region 1 with different sizes.](image)

As the terrain size increases, the total processing time is essentially determined by the external processing time. That seems to converge to about 80% and 70% of the total time when using 256 MB and 1 GB of RAM respectively.
We also compared EMVS to the IO\_VS algorithm of Haverkort et al. That is an adaptation for external processing of a method proposed by Van Kreveld [32] to compute the viewshed using a radial sweep of a terrain. Basically, Van Kreveld’s algorithm uses a plane sweep technique. Starting with a grid and a viewpoint \( p \), it rotates a sweep line around \( p \), computing the visibility of each cell in the terrain when the sweep-line passes over its center. It implements this with an active-structure data structure to contain the cells currently intersected by the sweep-line (the active cells). When a cell is intersected by the sweep-line, it is inserted in the active structure; when a cell stops being intersected by the sweep-line, it is deleted from the active structure. When the center of a cell is intersected by the sweep line, the active structure is queried to find out if that cell is visible. Thus, each cell in the grid has three associated events: when it is first intersected by the sweep-line and entered in the data structure, when the sweep-line passes over its center, and when it is last intersected by the sweep-line and removed from the data structure.

Haverkort et al. extended that into an algorithm to compute the viewshed on terrain stored in external memory, where the cells are sorted based on when they will be processed by the radial sweep.

Table 3 compares EMVS to the results reported for IO\_VS in [22], with the observer 1 meter above the terrain. The EMVS values were averaged from the three corresponding values for Regions 1, 2 and 3 listed in Tables 1 and 2. From Table 3, we see that EMVS is more than 6 times faster than IO\_VS. Further, IO\_VS was tested by its author on a Power Macintosh G5 dual 2.5 GHz, 1GB RAM and 80 GB 7200 RPM, which is significantly faster than the machine we used. Therefore, EMVS’s relative advantage is probably even greater. Finally, EMVS is much simpler to implement.
Why? EMVS uses a simple data structure — a sorted list. After the external sort, no list updates (insert or delete) are required. On the other hand, IO_VS uses more complex data structures, manipulated using recursive searching and updating.

<table>
<thead>
<tr>
<th>Terrain Size</th>
<th>EMVS</th>
<th>IO_VS</th>
</tr>
</thead>
<tbody>
<tr>
<td>119 MB</td>
<td>18</td>
<td>353</td>
</tr>
<tr>
<td>311 MB</td>
<td>46</td>
<td>865</td>
</tr>
<tr>
<td>1122 MB</td>
<td>209</td>
<td>3546</td>
</tr>
<tr>
<td>4264 MB</td>
<td>2627</td>
<td>16895</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Terrain Size</th>
<th>EMVS</th>
<th>IO_VS</th>
</tr>
</thead>
<tbody>
<tr>
<td>119 MB</td>
<td>22</td>
<td>364</td>
</tr>
<tr>
<td>311 MB</td>
<td>49</td>
<td>916</td>
</tr>
<tr>
<td>1122 MB</td>
<td>709</td>
<td>4831</td>
</tr>
<tr>
<td>4264 MB</td>
<td>5099</td>
<td>40734</td>
</tr>
</tbody>
</table>

Table 3: Execution time (seconds) comparison between EMVS and IO_VS for 1GB and 256 MB of RAM

8 Conclusions

We have presented EMVS, a very efficient algorithm to compute the viewshed of a point in a huge raster DEM terrain stored in external memory. EMVS is more than 6 times faster than the algorithm of Haverkort et al [22] and also, it can process very large datasets; we used it on a 6.1GB terrain. Finally, EMVS algorithm is quite simple to understand and to implement. It is available at Andrade [2] as an open source code distributed under Creative Common GNU GPL license [12].

Acknowledgment

This work was partially supported by CNPq - the Brazilian Council of Technological and Scientific Development, FAPEMIG - the Research Support Foundation of the State of Minas Gerais (Brazil) and by NSF grants CCR-0306502 and DMS-0327634 and by DARPA/DSO/GeoStar.
References


22


1. Introduction
We present an algorithm for efficient path planning on complex terrain with an arbitrary cost metric given by a 2D array corresponding to edge weights.

The agent is allowed the full range of Euclidean motion on the 2-dimensional plane, unlike alternate path planning schemes that strictly avoid obstacles, such as the Lee [3] and Hightower [2] algorithms or performing a graph search on the Voronoi diagram of the obstacle boundaries [1]. We use two runs of the A* algorithm to efficiently compute this path.

We assume that the agent has complete knowledge of the terrain. The agent’s goal is to plan a path between two given endpoints that minimizes a given cost metric. For example, the metric may penalize the agent for entering an area that is being observed by an opponent, or it may account for the slope of the terrain.

Our path finding routine is an adaptation of the A* algorithm. The first cost metric we evaluated was simply the number of grid points visited (similar to [3]). For the A* algorithm, the terrain is represented as an \( n \times n \) grid, each point on the terrain is a separate node, and each node has up to eight children in the search tree, corresponding to eight neighboring points (see Figure 1a). This method has the limitation of minimizing only the Chebyshev distance between the end points [4]. The Chebyshev distance between points \((x_1, y_1)\) and \((x_2, y_2)\) is defined as \(\max(|x_1-x_2|, |y_1-y_2|)\). Our implementation deviates slightly from the actual Chebyshev formula, in that the path length is computed via the Euclidean formula, but only movement in eight directions is considered while planning the path.

2. Chebyshev vs. Euclidean Distance
We start by considering a simple case, where many points on the terrain are obstacles and are marked as untraversable. The agent would like to take the shortest path through the terrain that avoids the obstacles. The cost metric is simply the total distance traversed. A naive method to allow for a full range of Euclidean motion, as shown in Figure 1b, would be to include edges between all grid point pairs in the search space. However, this increases the size of the search space from \(O(n^2)\) to \(O(n^3)\), as there are \(O(n^4)\) possible edges to consider. Also, to compute the true cost of each edge requires \(O(n)\) time, because the edge must be segmented as described below. This is clearly too expensive for larger terrains.

To speed up the algorithm, we designed a two-pass system. On the first pass, all points on the terrain are included in the search space, and the Chebyshev path is computed as described in the previous section. On the second pass, the only nodes in the search space are the points that are retained in the first path, and an edge is added to every other node in the search tree (see Figure 2). Thus, for any pair of points in the search space, the smuggler may traverse a straight line connecting them. In practice, computing this second pass is more efficient than the first pass.

Although our 2-pass algorithm is not guaranteed to be the optimal Euclidean path, the output from our heuristic 2-pass system does very well in practice. Our approach will never produce a path worse than the original Chebyshev path.
largest possible difference between the optimal Euclidean and Chebyshev paths will occur in a situation as in 3. The 135° angle is fixed since we are always using a regularly spaced grid. Then \((a + b) - c\) will be maximized when \(\theta = \phi = 22.5^\circ\). Therefore, the optimal Euclidean path should be no less than 92% of the length of the Chebyshev path. Thus, as our heuristic scheme cannot produce a path longer than the Chebyshev path, our heuristic algorithm should produce a path whose length differs by no more than 8% from the optimal Euclidean path. In practice, this difference is usually much less than 8%.

### 3. Results

For comparison, we compute the optimal Euclidean path with a brute-force application of the A* algorithm. Every pair of grid points, whether adjacent or not, is included in the search space. We compared our heuristic algorithm against the brute-force method by running both algorithms on a hundred 100 × 100 data sets. We limited the size of the datasets to 100 × 100, as the brute-force algorithm cannot efficiently handle larger datasets, though our heuristic approach runs quickly on 3200 × 3200 datasets. The average difference in the lengths of the computed paths was less than 0.1%, while the average speedup was greater than 100. Our heuristic approach is much faster than the brute-force scheme without significantly sacrificing the solution quality. Some sample results are shown in Figures 4 and 5.

### 4. Alternate Cost Functions

Our scheme can then be generalized to more sophisticated cost metrics. Rather than considering simply Euclidean distance, we are given a 2D array which corresponds to the costs of moving onto a grid point from any adjacent grid point. (The cost of moving from point A onto an adjacent point B need not be the same as the cost of moving from point B onto point A.) For computing the Chebyshev path, calculating the cost to move between adjacent points is trivial. However, for the second pass which allows a full range of Euclidean motion, the cost to traverse a straight line that connects two distant points must be computed. This line is not likely to pass through grid points exactly. Here the cost metric at several places (not necessarily at grid points) along the line must be interpolated. All the points that lie along gridlines are used, and each point is linearly interpolated from its two closest grid points.

Now our path planning procedure takes a cost function defined on a uniform grid, with non-uniform edge weights, and computes the path that minimizes the cost function while allowing a full range of Euclidean motion.

### 5. Acknowledgements

This research was supported by NSF grants CCR-0306502 and DMS-0327634, by DARPA/DSO/GeoStar, and by CNPq - the Brazilian Council of Technological and Scientific Development.

### 6. References


Path Planning on a Compressed Terrain

Daniel M. Tracy, W. Randolph Franklin, Barbara Cutler, Franklin T. Luk, Marcus Andrade, Jared Stookey
Rensselaer Polytechnic Institute

Motivation

- Terrain representation
- Smugglers and border guards
Terrain Compression

- Must evaluate the information loss of the compression
- Reconstitute the terrain from the compressed data to obtain the alternate representation
- Compare the alternate representation against the original
- Simple metrics such as RMS and max elevation error
- More complex metrics such as visibility and path planning

Outline

- New path planning algorithm
  - Account for complex cost metric
  - Allow for full range of Euclidean motion on a 2D grid
  - Efficient on hi-res data
- Novel error metrics to evaluate terrain compression

October 31, 2008
Siting & Path Planning

- Border guard placement: Multiple Observer Siting
- Smuggler’s Path: Find the shortest path between two given points while trying to avoid detection by the observers.
- A* algorithm
- Add penalty for going uphill.

Cost Metric

- Cost of moving from one cell to an adjacent cell:
  \[ \text{Cost} = \sqrt{(h^2 + v^2)} \times \text{SlopePenalty} \times \text{VisibilityPenalty} \]
- \( h \) is the horizontal distance.
- \( v \) is the elevation difference.
- SlopePenalty is \( 1 + \frac{v}{h} \) when going uphill and 1 otherwise.
- VisibilityPenalty is 1 if the new cell is not visible and 100 otherwise.
Range of Motion

Chebyshev  Euclidean

A straightforward application of the A* algorithm results in the Chebyshev distance being minimized, rather than the Euclidean distance.

October 31, 2008

Path Planning

- New approach: Two pass system
- First pass: Plan a path that minimizes Chebyshev distance.
- Second pass: Only include points from the first path in the search space.
- Not guaranteed to be optimal, but in practice it often is.

October 31, 2008
Brute Force Comparison

Chebyshev  Heuristic  Brute Force

100 100x100 test cases
• Average path length difference of 0.1%
• Average speed up of over 100.

BruteForce ≤ Heuristic ≤ Chebyshev

October 31, 2008

RPI Geo* Final Report 243/919  FWC 2008 Talk 1
Test Data
(400x400 DTED II)

W111
N31
subsets

Hill1
Hill2
Hill3

W121
N38
subsets

Mtn1
Mtn2
Mtn3

October 31, 2008

Error Metrics

Path Cost Error: Difference of the costs of the paths computed on the original and alternate representations.

Alternate
Original

October 31, 2008
Hill 3
Elevation range: 500 m
Elevation stddev: 59 m

Original  October 31, 2008  Alternate

Mtn 1
Elevation range: 1040 m
Elevation stddev: 146 m

Original  October 31, 2008  Alternate
Mtn 2
Elevation range: 953 m
Elevation stdev: 152 m

Original  October 31, 2008  Alternate

Ottawa LIDAR Data
- 2000x2000 grid
- 19 minutes on 2.4 GHz CPU with 4 GB memory
- peak memory usage 360 MB

October 31, 2008
Multiple Queries

- Sample a larger portion of the terrain by performing multiple path planning queries

Future Work

- Scale visibility penalty by distance from observer
- Make sure that the hidden areas are disconnected
- Moving observers: Compute paths for tourists, smugglers
- Red/Blue games: The blue team tries to hide; the red team tries to find them
Summary

Path Planning Algorithm
- Accounts for complex cost metrics
- Full range of Euclidean motion
- Efficient on hi-res terrain
- New error metrics derived from smugglers and border guards for evaluating terrain compression.
Compressing Terrain Slopes with ODETLAP

Daniel M. Tracy, W. Randolph Franklin, Barbara Cutler, Marcus Andrade, Metin Inanc, Zhongyi Xie
Rensselaer Polytechnic Institute

Abstract:
We extend the ODETLAP compression algorithm to include slope equations to specifically target the compression of terrain slopes. Given a subset of the elevations and slopes from the terrain, ODETLAP can reconstruct a full-resolution approximation of the terrain.

Point Selection:
Original Terrain Representation → Initial Elevations Selection → Reconstructed Surface

Slope Errors → Error Small Enough? → Select Additional Slope Values

ODETLAP:
Overdetermined set of linear equations
Original ODETLAP equations:
Discrete approx. of Laplacian PDE:
\[ z_{i,j} = \frac{z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1}}{4} \]
Known elevations:
\[ z_{i,j} = h_{i,j} \]
Plus new slope equations, derived from Zevenbergen-Thorne slope method:
\[ z_{i+1,j} - z_{i-1,j} = h_{i+1,j} - h_{i-1,j} \]
\[ z_{i,j+1} - z_{i,j-1} = h_{i,j+1} - h_{i,j-1} \]

Algorithm:
1. Start with regular grid of elevations.
2. Iteratively solve ODETLAP and add points with largest slope error to the system.
3. Encode (x,y) with Run Length Coding.
4. Encode Δz lossily with k-means clustering.

Compression vs. error (400x400 terrain)

References:
1. W. Randolph Franklin, Metin Inanc, and Zhongyi Xie, “Two Novel Surface Representation Techniques”, Autocarto 2006
Operating on Large Geometric Datasets

W. Randolph Franklin
Rensselaer Polytechnic Institute
ECSE Dept, 110 8th Street, Troy, NY 12180, USA
frankwr@rpi.edu, http://wrfranklin.org/

1. INTRODUCTION

We describe some successful strategies for storing and operating on sets of up to tens of millions of simple geometric objects. Users wish to process datasets derived from laser scanners that may range up petabytes in size. At this point on the asymptotic time curve, superlinear times, even \( T = \theta(N \log N) \), are too slow. We illustrate this with some implementations.

2. MINIMIZE THE EXPLICIT TOPOLOGY

A data structure that explicitly stores the minimum possible structure will be more compact. Surprisingly, this may also lead to simpler algorithms. For example, how simple can a polyhedron \( P \) be? Consider the set of faces, \( F = \{ f_i \} \). That permits the following operations.

1. Inclusion determination: Point \( x \) is contained in \( P \) iff a semi-infinite ray from \( x \) crosses an odd number of \( f_i \).
2. Volume computation: The volume \( V \) of \( P \) is the sum of the volumes of all the pyramids determined by the \( f_i \) and the origin. Other mass properties follow similarly.

In no case was the global topology of the hierarchy of nested inclusions of shells of faces required, although it could have been derived if necessary.

However, a simpler data structure is possible. Suppose that we have only the set of incidences of vertices, edges lines, and oriented face planes. For instance, for a cube, each vertex would induce six such incidences. Then if \( P \) is \( \{ v \} \), \( \hat{v} \) is a unit tangent vector along the edge incident on it, \( N \) is a unit vector normal to \( \hat{v} \) in the plane of the face, and \( \hat{B} \) is a unit bimodal vector, normal to both \( \hat{v} \) and \( N \), pointing into the polyhedron, then the volume is

\[
V = -\frac{1}{2} \sum P \cdot \hat{T} \cdot N \cdot P \cdot \hat{B}.
\]

Similar formulae obtain for other mass properties. Figure 1 illustrates this for a cube. Each visible (vertex, edge, face) tuple is checked marked, except for one that is starred. For that one, the vectors \( P, T, N, \) and \( \hat{B} \) are shown.

3. DESIGN FOR EXPECTED INPUT

We find pairs of coinciding input objects with a uniform grid, which is simple to implement and fast to execute. Although the worst-case time is \( \theta(N^2) \) or \( \theta(N^3) \), the expected time is \( \theta(N) \).

4. SHORT-CIRCUIT OPERATOR COMPOSITIONS

E.g., computing the volume of the union \( C \) of two polyhedra \( A \) and \( B \) does not require completely computing \( C \). From Section 2, computing certain local information of \( C \) suffices. This principle becomes even more important with more complex operations. Suppose that we want the volume of the union of \( N \) polyhedra? The naive algorithm proceeds by unifying the polyhedra pair by pair, then four by four, and so on, building a tree of depth \( \log N \). To explicitly compute the union polyhedron before finding its volume. Worse, the intermediate polyhedron may have many more vertices than the final one. However, that whole tree may be flattened as described in the following section.

5. VOLUME OF THE UNION OF MANY POLYHEDRA

This is an algorithm to compute the volume of the union of many polyhedra in expected linear time. Algorithm details are in [3] and the time analysis in [2]. We have implemented and tested this for up to \( 3 \cdot 10^7 \) identical isothetic cubes.

1. Superimpose a uniform grid on the input data. A good cell size is \( 1/2 \) the cube size.
2. Iterate through the input cubes. Mark any grid cell that is completely contained in a cube as covered. Record every non-covered cell that an input vertex, edge, face, or cube intersects, in a data structure indexed by the cell.
3. Initialize variables \( V, A \) and \( L \) to zero. They will accumulate the total volume, surface area, and edge length of the union of the input cubes.
4. Iterate through the cells. In each cell, find all intersections of an edge and a face, or of three faces. These are potential output vertices. Other potential output vertices are the input vertices.
5. Test each potential output vertex against the input cubes in the same call, to cull those contained in any cube. The rest are the output vertices.
6. Knowing each output vertex’s neighborhood from how it was formed, compute its contribution to \( V, A \) and \( L \) and update them.

5.1 Implementation Tests

The HW is a dual 2.4 GHz Xeon with 4Gib of real memory. The SW is SuSE 8.2 linux and the Intel C++ compiler. The program is about 1000 lines of code, excluding debugging lines, comments, and blank lines. The input cubes are generated with a combination of three Tausworth random...
number generators, which is much better than the widely used class of linear congruential generators. One problem with even the best linear congruential generators is that, if we let the generated numbers be \(x_i\), then the 3-D points \((x_i, x_{i+1}, x_{i+2})\) fall on a relatively small number of parallel planes.

Figure 2 shows some sample runs, varying the number of cubes, edge length, grid resolution, and number of processors. The grid resolution and number of processors affect the time for a given input. For each run, the CPU time is reported, conservatively, as the sum over all the processes. However, since 4 threads can execute in parallel, the elapsed time is usually much less.

The added line has a slope of one, which would be \(T = \Theta(N)\). The observed time performance appears to be slightly worse than linear. This is principally caused by memory limitations, which force a suboptimally small grid resolution to be used for large datasets.

6. COMPUTING THE AREAS OF THE NON-EMPTY INTERSECTIONS OF PAIRS OF CELLS FROM TWO OVERLAPPING TRIANGULATIONS

Consider two different triangulations, \(T_1\) and \(T_2\), over the same region. In \(E^2\), for the United States, \(T_1\) might be counties and \(T_2\) hydrography regions. A polygon \(p\) of \(T_1\) is generally not contained in any one polygon \(q\) of \(T_2\), but overlaps several \(q_i\). Suppose that we know the populations of the \(p_i\), and wish to infer the populations of the \(q_i\). One way to estimate \(\text{pop}(p_i)\) is to pro-rate the populations of the \(p_i\) overlapping \(q_i\), each weighted by the fraction of \(\text{area}(q_i)\) that overlaps each \(p_i\). This \(\text{cross-area}\) problem requires determining all the pairs \((p_i, q_j)\) with non-empty intersections, and the areas of those intersections.

The first optimization is not to work with individual polygons, but with the planar graph as a whole. The second is to realize that computing the area of the intersection of two polygons requires only the intersecting polygons’ set of vertices together with their local topologies. That motivates the following algorithm.

1. Assume that for \(T_1\) and \(T_2\), we know each vertex’s position, and each edge’s adjacent vertices and polygons. No global information will be used.
2. Initialize a hash table \(\mathcal{H}\) keyed by \((i, j)\) with an entry for each \((p_i, q_j)\) with non-zero intersection, and contains the partial or complete area of the corresponding polygon.
3. Computing the area of an intersection requires computing its vertices, and each vertex’s local neighborhood, i.e., the directions of the two adjacent edges on it. These output vertices are input vertices or intersections of input edges.
4. Find all the intersections \(s\) between edges of \(T_1\) and \(T_2\) using a uniform grid. Each \(s\) is a vertex of four output intersection polygons. Compute \(s\)’s contribution to those polygons’ areas and create or update four entries in \(\mathcal{H}\).
5. Process all the input vertices similarly.
6. The non-empty entries of \(\mathcal{H}\) are the desired information.

An implementation in \(E^2\) is described in [5], with code at [1]. The \(E^3\) algorithm is described in [4]. The expected execution time, assuming i.i.d. input, is linear in the size of the input plus output, with the computation being quicker than the I/O. \(T_1\), coterminous US counties has 55068 vertices, 46116 edges, and 2985 polygons, while \(T_2\), hydrography polygons, has 76215 vertices, 60835 edges, 2075 polygons. The total time (excluding I/O) is 1.58 CPU seconds on a machine that is several years old.

7. ACKNOWLEDGEMENTS

This research was supported by DARPA/DSO/GeoStar.

8. REFERENCES

Operating on Large Geometric Datasets
FWCG2008

W. Randolph Franklin\(^1\)

Rensselaer Polytechnic Institute
Troy, NY, 12180

1 Nov, 2008

\(^1\)(518) 276-6077, frankwr@rpi.edu, http://wrfranklin.org/
The Problem and the conventional solution

Geometric datasets grow too large to easily fit into core.

Must use external algorithms.

1 I/O is expensive.
2 Random I/O is very expensive.
My approach

1. Stay in core as long as possible — fast, random.
2. Use compact data structures.
3. Minimize explicit topological data structures.
4. Use input-sensitive algorithms.
5. Optimize operator compositions.
6. Try for Linear time — superlinear times, even $T = \theta(N \log N)$, are too slow.

$x \log(x) > x$
Minimize the explicit topology

1. Explicitly storing the minimum possible structure saves space
2. and often facilitates simpler algorithms.
3. E.g., how simple can a polyhedron \( P \) be?
4. The set of faces, \( \mathcal{F} = \{f_i\} \), often suffices.
5. That permits
   1. **Inclusion determination:** Point \( x \) is contained in \( P \) iff a semi-infinite ray from \( x \) crosses an odd number of \( f_i \).
   2. **Volume computation:** The volume \( V \) of \( P \) is the sum of the volumes of all the pyramids determined by the \( f_i \) and the origin. Other mass properties follow similarly.
6. Global topology of the hierarchy of nested inclusions of shells of faces is never required
7. That could have been derived if necessary.
8. We can get even simpler.
Set of Incidences

1. Polyhedron: \( \{(P, \hat{T}, \hat{N}, \hat{B})\} \)
2. One per incidence, 6 per cube vertex.
3. \( V = -\frac{1}{6} \sum (P \cdot \hat{T}) (P \cdot \hat{N}) (P \cdot \hat{B}) \)
4. That's a Google map reduce.
5. Irrelevant: multiple nested components, nonmanifold vertices.
Input-sensitive

1. **Line segment intersections** in $E^2$
   - $K \triangleq $ number of intersections among $N$ line segments of length $L$ in $E^2$ $1 \times 1$ region.
   - $K_{\text{max}} = N^2 / 2$
   - i.i.d input: $\overline{K} = N^2 L^2 / 4$

2. **Visible edge intersections** among overlaid squares
   - Overlay, 1-by-1, $N \times L \times L$ random squares inside $1 \times 1$ region.
   - Later squares hide earlier squares.
   - $K \triangleq $ number of edge intersections
   - i.i.d input: $\overline{K} = \theta(N^2 L^2)$
   - $K_v \triangleq $ number of visible edge intersections
   - $K_{v,\text{max}} = N^2 / 2$
   - i.i.d input: $\overline{K}_v = \theta(N)$
   - independent of depth ($N L^2$) of scene.
Time for visible edge intersections among overlaid squares

1 This linearity is key to linearity of volume of cube union later.

2 \( N \triangleq \) number of squares, \( L \triangleq \) edge length, \( G \triangleq \) number of grid cells per side = \( 3/L \)

3 \cdots \) tedious derivation \cdots

4 expected number of intersection tests performed per cell is

\[ N_{tpc} \leq 2L^2 N \left( 1 + 4L^2 N \right) e^{-\frac{L^2 N}{4}} \]

5 Total number of intersection tests

\[ N_{tt} = 9L^{-2} N_{tpc} = 18N \left( 1 + 4L^2 N \right) e^{-\frac{L^2 N}{4}} < 18\left( 1 + 16e^{-1} \right) N \]
Segment intersection is linear time for every non i.i.d application we’ve tried

Roads

Counties, hydrog

VLSI

Nonuniform mesh
Optimize operator compositions

1 Goal\textsubscript{1} Volume of union of two polyhedra
2 Do not first compute union.
3 Requires only set of vertices of result, with their neighborhoods.
4 Does not require any global info. \textit{No edges. No faces.}

1 Goal\textsubscript{2} Volume of union of many polyhedra
2 Requires only \ldots
3 Does not require \ldots
4 Does not require Building a computation tree of depth \textup{\(\log(N)\)}
5 No intermediate swell.
Volume of the union of many cubes

Union of cubes

Don’t need this computation

tree

Output vertices

Culling possible intersections
Volume of the union of many cubes

1. \[ V = \sum s_i x_i y_i z_i \]
2. This is exact, not Monte Carlo.
3. Output vertex is input vertex, or union of input face and edge, or union of 3 faces.
4. Output vertex is not in any input cube.
5. Determine neighborhood of each output vertex.
6. Superimpose uniform grid, proportional to cube size
7. In a cell contained in one cube, no output vertices in that cell.
8. Number of surviving vertices is linear.
9. Linear time.
Volume of the union of many cubes — implementation

Number of cubes vs time

1 < 1000 lines of C++ on 2.4GHz dual Xeon.

2 Input: Up to 30 000 000 identical isothetic random cubes.
Conclusion

1. Simple, local topological, data structures
2. Optimizing composition of operators facilitate
   1. linear expected time algorithms
   2. for processing large data structures
Parallel ODETLAP for Terrain Compression and Reconstruction

Jared Stookey
ECSE Dept, RPI
jstookey@jstookey.com

Zhongyi Xie
CS Dept, RPI
xiez@rpi.edu

Barbara Cutler
CS Dept, RPI
cutler@cs.rpi.edu

W. Randolph Franklin
ECSE Dept, RPI
frankwr@rpi.edu

Dan Tracy
CS Dept, RPI
tracyd@rpi.edu

Marcus V. A. Andrade
DPI - Univ. Fed. Vicosa -
Brazil and RPI
marcus.ufv@gmail.com

ABSTRACT
We introduce a parallel approximation of an Over-determined Laplacian Partial Differential Equation solver (ODETLAP) applied to the compression and restoration of terrain data used for Geographical Information Systems (GIS). ODETLAP can be used to reconstruct a compressed elevation map, or to generate a dense regular grid from airborne Light Detection and Ranging (LIDAR) point cloud data. With previous methods, the time to execute ODETLAP does not scale well with the size of the input elevation map, resulting in running times that are prohibitively long for large data sets. Our algorithm divides the data set into patches, runs ODETLAP on each patch, and then merges the patches together. This method gives two distinct speed improvements. First, we provide scalability by reducing the complexity such that the execution time grows almost linearly with the size of the input, even when run on a single processor. Second, we are able to calculate ODETLAP on the patches concurrently in a parallel or distributed environment. Our new patch-based implementation takes 2 seconds to run ODETLAP on an 800 x 800 elevation map using 128 processors, while the original version of ODETLAP takes nearly 10 minutes on a single processor (271 times longer). We demonstrate the effectiveness of the new algorithm by running it on data sets as large as 16000 x 16000 on a cluster of computers. We also discuss our preliminary results from running on an IBM Blue Gene/L system with 32,768 processors.

Categories and Subject Descriptors
I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling

Keywords
GIS, LIDAR, PDE solver, parallel computation, terrain modeling, terrain elevation data set compression, terrain interpolation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.


1. INTRODUCTION
Due to recent advances in GIS data acquisition methods such as LIDAR and satellite imagery, vast amounts of GIS data is being collected at such a fast rate that storing and processing all of this data currently poses a problem, and will increasingly pose a problem in the near future. We can help to reduce these problems in several ways. First, we can reduce the size of the stored data by inventing new compression techniques that are optimized for GIS data. Second, we can process data more quickly using new approximation techniques that will reduce the time required for operating on larger data sets. Third, by taking advantage of machines such as multi-processor and multi-core machines, as well as computing clusters and supercomputers, we will be able to transform vast amounts of raw data collected in the field into strategic planning intelligence in a short amount of time. In this paper, we introduce a method to accelerate the process of ODETLAP terrain compression and reconstruction [13] that demonstrates each of these.

ODETLAP has proven to be an effective method for generating data that closely represents the original terrain by filling in the unknown points. This feature can be used to generate raster data from sparse point cloud data, or to reconstruct an elevation map from a small subset of points. Using ODETLAP as a black box, we have developed terrain compression algorithms to reduce the storage space for large terrains. However, ODETLAP suffers from a scalabil-
We apply ODETLAP to overlapping layers of patches. This example shows a 400x400 grid with 49 100x100 patches. In this example, 16 patches would be enough to cover the entire heightmap without any overlap, but we would see errors at the patch edges. Instead, we run ODETLAP on 49 overlapping patches.

As the dimensions of the elevation map increase, the numerical complexity of ODETLAP causes the computation time to increase quadratically with the number of pixels (Figure 5). In this paper, we describe an approximation for ODETLAP in which we divide the terrain into overlapping patches which ODETLAP can process quickly (Figure 2), and later merge the results together as shown in Figure 3. This method, which we call patch-based ODETLAP, quantizes the numerical complexity of ODETLAP, so that the execution time grows linearly with the size of the input matrix. Also, by dividing the input matrix into separate patches on which ODETLAP is run, we are able to take advantage of parallel and distributed systems to calculate multiple patches simultaneously. By combining both of these advantages, we are able to gain tremendous speed improvements for large terrain maps when compared to the original version, which we call non-patch ODETLAP.

Figure 1 provides a powerful demonstration of ODETLAP’s ability to fill in unknown points. Given over 13 million sparse point cloud data points generated using airborne LIDAR, ODETLAP generates a very accurate representation of the original urban elevation map. Using our parallel patch-based ODETLAP, this calculation took 33 minutes and 22 seconds on 128 processors in a cluster of 2.6 GHz AMD Opterons. Extrapolating from the data in Figure 5, the same calculation would have taken more than 179 days using the non-patch version of ODETLAP on a single processor.

Parallel ODETLAP greatly improves the efficiency and still compares favorably with the non-patch ODETLAP in reconstruction accuracy. Because we provide generous overlap between the patches and ensure that sufficient known data samples are shared between the patches, the result of the patch-based parallel ODETLAP matches the non-patch version of ODETLAP within 0.1%.

Our contributions are as follows:

- A partitioning scheme for calculating ODETLAP that greatly reduces the overall numerical complexity for large elevation maps.
- A method for distributing and merging data partitions in order to calculate ODETLAP in a parallel or distributed environment.
- Demonstration of our algorithm for calculating ODETLAP on much larger terrains than what was previously possible with nearly identical results.

2. RELATED WORK

Research on parallel computing in GIS began to play an important role in the 1990s [2] as parallel computing technology became more widely available. Research has been done on parallelizing existing algorithms like line simplification by Mower [10], polygon line shading by Roche et al [11] and processing heterogeneous networks for GIS by Clematis [1]. Research at the Edinburgh Parallel Computing Centre (EPCC) emphasized creating new parallel libraries to support high performance GIS data models [2, 6, 9].

A newer approach to the problem of parallelizing GIS operations was taken in [7] in which the GRASS GIS application was modified to operate in a clustered environment. Ichikawa et al [12] demonstrate an iterative data partitioning scheme for parallelizing a PDE solver. Griebel et al [5] made excellent progress using parallel multigrid to solve PDE’s. More recently, we have seen research in image processing that involves large linear systems on huge images using a streaming multigrid that can benefit by running on parallel systems [8] in such a way that could be adapted to problems in GIS.

We extend these important pieces of research to overdetermined PDE’s. We specifically target GIS applications and provide an in-depth analysis of the quality-cost trade-off associated with partitioning very large elevation maps.
In this paper, we propose a parallel terrain compression algorithm that is capable of processing terrain maps of size 16000 × 16000 and larger in a reasonable amount of time.

2.1 The ODETLAP Solver

The Over-determined Laplacian Approximation (ODETLAP), [13] is an extension to the Laplacian equation:

\[ 4z_{xy} = z_{x-1,y} + z_{x+1,y} + z_{x,y-1} + z_{x,y+1} \] (1)

This equation states that for every non-border point identified by coordinate \((x, y)\) in the elevation matrix, the elevation \(z_{xy}\) is equal to the average of its neighbors. The handling of border points forms a special case and is omitted due to the lack of deep theoretical interest. This equation by itself is unable to represent local maxima in terrain modeling [4], thus we also include a second equation:

\[ z_{xy} = h_{xy} \] (2)

Equations 1 and 2 form a basis for our over-determined linear system. The system’s input is a set of points \((x, y, z)\) which specify the elevation of certain locations \((x, y)\). For those locations with known elevation, we have both equations, and for the rest of the locations, only equation 1 is specified. The relative importance of these two sets of equations is determined by a parameter \(R\) during the interpolation process. Weighting equation 2 over equation 1 results in a more accurate surface which sacrifices smoothness, while weighting equation 1 over equation 2 gives us a smoother surface that interpolates the known points.

We use ODETLAP when we have incomplete information about the actual elevation matrix. The known value and the Laplacian constraint (the average of its neighbors) can be used to interpolate the elevation value for every unknown and known point. In this way, ODETLAP can be considered as a solver whose input is a set of known points \((x, y, z)\) and an interpolation parameter \(R\) and output is the Digital Elevation Model (DEM) matrix of the complete terrain. ODETLAP has several benefits that are ideal for terrain data, including the ability to handle continuous as well as broken contour lines of elevations, processing kidney-bean-shaped contours without giving fictitious results at regions inside, and the ability to infer local maxima from a series of contours.

2.2 ODETLAP-based Terrain Compression

Since ODETLAP is capable of reconstructing the whole DEM matrix from a few sparse input points \((x, y, z)\), it can be used as a decompressor in the ODETLAP-based compression algorithm. In order to reduce the amount of space required to store the data, only a limited subset of the elevations from the original elevation data are stored. Later, ODETLAP is used to losslessly reconstruct the elevation map by filling in the missing points.

Figure 4 presents the flowchart of the algorithm [4]. The DEM first undergoes a point selection which picks a subset of posts, \(S\), as input to the ODETLAP solver. ODETLAP selects points that are deemed important to the accurate reconstruction of the terrain such as contour lines, border points, or any other available points. The points can consist of a sparse point cloud, a regular grid, or a mix between the two. The ODETLAP solver reconstructs from \(S\) the whole DEM matrix of elevations giving us an initial approximation of the elevation matrix. We initially pick a very small subset of points (\(|S| \leq 1000\) in a 400 × 400 elevation matrix), yielding an approximation with insufficient accuracy. After obtaining the initial approximation, iterative refinement is performed to insert additional elevation values where the reconstructed surface most poorly matches the original data. The refinement steps end when the overall RMS elevation error is below the specified threshold. Table 1 summarizes the ODETLAP algorithm’s compression performance on three mountainous terrain samples of size 400 × 400.

![Figure 4: ODETLAP algorithm: Square boxes represent data and curved boxes (light green) represent operations.](image)

<table>
<thead>
<tr>
<th></th>
<th>Mtn1</th>
<th>Mtn2</th>
<th>Mtn3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elev range</td>
<td>1040m</td>
<td>955m</td>
<td>788m</td>
</tr>
<tr>
<td>St. Dev.</td>
<td>146.0</td>
<td>152.4</td>
<td>160.7</td>
</tr>
<tr>
<td>Orig size</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
</tr>
<tr>
<td>Compr. size</td>
<td>9744B</td>
<td>9670B</td>
<td>9895B</td>
</tr>
<tr>
<td>Compr. ratio</td>
<td>33:1</td>
<td>33:1</td>
<td>32:1</td>
</tr>
<tr>
<td># pts selected</td>
<td>4160</td>
<td>4160</td>
<td>4160</td>
</tr>
<tr>
<td>Elev. RMS error</td>
<td>9.48</td>
<td>9.55</td>
<td>9.68</td>
</tr>
<tr>
<td>Elev. RMS perc.</td>
<td>0.91%</td>
<td>1%</td>
<td>1.23%</td>
</tr>
<tr>
<td>Slope RMS error</td>
<td>8.34°</td>
<td>8.36°</td>
<td>7.87°</td>
</tr>
</tbody>
</table>

Table 1: Results from compressing three mountainous terrain data sets using non-patch ODETLAP.

2.3 Regular and Irregular Point Data

In this paper we demonstrate ODETLAP in two different reconstruction scenarios. In Figure 1 a LIDAR point cloud with approximately 13 million points is inflated to a 100 million point dense grid. In the other examples in the paper we reconstruct the terrain from a compressed subset of points consisting of 1% of the total points, uniformly sampled in a regular grid. For example, a 400 × 400 grid consisting of 160000 points will be represented by 1600 values in its compressed form. With the ODETLAP compression scheme we can optionally augment the regular grid with additional important points to reduce the error in the reconstructed elevation map, however we do not concern ourselves with this in the context of this paper because we are focusing on the
parallelization of the algorithm rather than optimizing the compression method. The primary difficulty for using only irregular data with the patch method is ensuring that every patch contains enough data and sufficiently overlaps the data from neighboring patches to accurately reconstruct the unknown data points. We leave this as an area for future work.

3. OUR APPROACH

The time it takes to calculate ODETLAP on an elevation map does not scale as the size of the data set increases, and the execution time quickly becomes prohibitively long. Figure 5 illustrates the ODETLAP performance issue. When the terrain is of size $50 \times 50$ and $100 \times 100$, it takes less than one second to run ODETLAP. However, when we run ODETLAP on an $800 \times 800$ terrain, it takes nearly 10 minutes to finish, and $800 \times 800$ is a very modest terrain size. Thus, our goal is to develop a scalable implementation of ODETLAP to allow manipulation of large terrains, such as those collected by LIDAR scanners.

As the data set size grows, more and more points are involved in the calculation, some of which are quite distant and thus have little influence on each other. In Figure 6, we show that when we edit a single known point in the input to ODETLAP, only points within a small neighborhood of the point are affected. Beyond that small region, the effect becomes negligible. This supports our hypothesis that it should be possible to divide large data sets into separate patches, run ODETLAP on them individually, and achieve similar results to the non-patch ODETLAP solution. Specifically, a patch size of $100 \times 100$ should be sufficient to capture the detail for a terrain grid with a subsampling percentage of 1%. When using a patch size of $100 \times 100$, each patch will overlap with a window of $100 \times 50$ with its neighbors to the north and south, $50 \times 100$ with its neighbors to the east and west, and $25 \times 25$ with its diagonal neighbors. If we used a higher subsampling percentage, then less overlap would be needed, so a smaller patch size would be appropriate. When the reconstructed patches are merged back together into the reconstructed elevation map as shown in Figure 7, the difference between the reconstructed and original should be minimal.

3.1 Dividing into Patches

The ODETLAP calculation depends on elevation data from neighboring pixels, which causes a problem when running it on individual patches. At the edges of a patch, there is less information to work with, and therefore the calculations tend to have errors when compared to the non-patch-based ODETLAP solution. Figure 8 highlights the problem by showing that when the image is reconstructed, the values are generally correct near the center of the patch, but near the edges of the patch, it is common to encounter errors of 5 meters or more.

To determine the value for the error, we calculate ODETLAP on the heightmap using a single large patch that covers the entire elevation map, and then we calculate ODETLAP on each individual patch. The error is defined as the absolute value of the difference between the corresponding elevation values for each method. If we were to simply divide a terrain map into a single layer of non-overlapping patches, then reconstruct them with ODETLAP, and join them at their edges to form the reconstructed image, then we would see results like the ones in Figure 9. At the patch borders, there are areas of large error and drastic discontinuities.

To avoid errors at the borders of the patches, we run ODETLAP on overlapping layers of patches. Figure 2 illustrates an example where instead of dividing an elevation map into 49 overlapping patches, we divide the elevation map into 16 non-overlapping patches, and then we calculate ODETLAP on 31 overlapping layers of patches. At the patch borders, there are areas of large error and drastic discontinuities.

3.2 Merging the Patches

After ODETLAP is calculated on multiple patches, the
In the patch ODETLAP method, we take the compressed terrain (A), and divide it into patches (B). Next, we run ODETLAP on each patch individually, which reconstructs a small portion of the entire elevation map (C). Finally, we merge all of the patches into the final approximated solution (D).

Due to incomplete data, ODETLAP results in errors near the borders of a patch when we compare the results from the non-patch ODETLAP method with the results from running ODETLAP on a small patch. The error plot on the right shows correct results in blue, and elevation differences of 5 or more units (meters) in red.

In order to accomplish this, we use bilinear interpolation which weights pixels near the center very strongly, and the weight falls off to zero for pixels near the edges. A visualization of the weights can be seen in Figure 11. The image on the left shows the bilinear interpolation weighting pattern for a single patch. The solid green image on the right shows the weighting pattern using simple averaging. Blue pixels represent a weight of 0.0, and red pixels represent a weight of 1.0. In both weighting methods, when four patches are merged, the sum of all of the weights for a given pixel is one. Using bilinear interpolation, we are able to merge the patches such that the pixels at the patch’s borders are ignored, but the correct pixels near the center contribute most of the value. This results in a reconstructed elevation map that has very small errors, and no visible discontinuities, as seen in Figure 12.

Because non-patch ODETLAP is prohibitively slow on large data sets, the error analysis and plots shown in Figures 8, 9, 10, and 12 were performed on 400 × 400 terrain data. The elevations for this DEM range from 1105 to 1610 meters, and are represented as integer values with units of one meter.

4. IMPLEMENTATION

We implemented the patch version of ODETLAP using MPI, which allows us to run the software on parallel and distributed platforms. When MPI starts, the first process is assigned the task of waiting for reconstructed patch data from the rest of the processes, which are designated as worker processes. All of the workers are pre-assigned a set of patches to process. For each assigned patch, the process does the following:

- Load the patch
- Run ODETLAP on the patch
- Send the reconstructed patch to the central process
We use bilinear interpolation (left) instead of a simple averaging (right) to do a weighted averaging of four pixels to merge four patches. Note that the corners and edges form special cases where only one and two patches contribute to the result.

We use bilinear interpolation to do a weighted average such that border values fall off to zero. This results in a visibly continuous reconstructed image (A), and small error values (B), when compared with results from running the non-patch version of ODETLAP on the same terrain.

As the patches are collected by the first process, the values are weighted and merged into the full reconstructed elevation map, which is then saved to the hard disk.

A simple direct linear system solver would have required cubic time to execute. However, in our implementation we built ODETLAP on the QR solver from the CSparse library [3]. We have found the execution time to be nearly quadratic with respect to the number of points in a single patch, and in the case of the non-patch ODETLAP version, with respect to the number of points in the entire input elevation map.

We implemented the parallelized patch ODETLAP on a cluster of 2.6 GHz AMD Opteron machines running Red Hat Enterprise Linux 4.5. We have also tested the program on the Blue Gene/L supercomputer. The Blue Gene presents additional challenges because each processor can optimally access only 512MB of memory. Also, running ODETLAP on a very large number of processors requires extra care to ensure that the slowness of disk access does not cause bottlenecks that prevent the worker processes from working at full capacity. We will discuss the Blue Gene further in Section 6.

5. RESULTS

We tested our parallelized ODETLAP on a 16000 × 16000 DTED Level 2 data set covering roughly 400,000 square miles primarily occupied by the states of Kansas and Nebraska in the USA. The terrain is divided into 101761 patches, each with a resolution of 100 × 100. We used 128 processors on a cluster of 2.6 GHz AMD Opterons and the entire computation took 28 minutes and 32 seconds. The full terrain is shown in Figure 13. The range (highest minus lowest) of the test data is 1013 meters and the standard deviation is 217 meters. We select every 10th sample point in both the x and y dimensions to generate a set of input points consisting of 1% of the total points. A single 100 × 100 patch consisting of 10000 points is represented by 100 points.

Compared to the original terrain, the reconstructed terrain has mean absolute error of 1.96, max absolute error of 50, and root mean square error of 2.76. Note that it is impossible to run the non-patch version of ODETLAP on this large data set, so for these results we are comparing to the original. A comparison of the original and reconstructed terrain is given in Figures 14, 15, and 16. The two images in Figure 14 correspond to the 1000 × 1000 patch in the top left corner of the original terrain in Figure 13. We can see in the figure that the reconstructed terrain is only losing some high frequency details. A difference map between the original and the result (Figure 15) shows which areas perform well, and where smoothing occurs. Figure 16 presents a close-up view of the area with the highest error, taken from the oxbow river flowing from the southeast corner to the center of Figure 13. The largest errors occur due to the presence of many extreme variations in elevation within a small (100 × 100) area. Many of these details are captured with the more advanced point selection schemes described.
Figure 14: The original terrain (A) is compared to terrain that has been compressed by sampling on a regular grid, and then reconstructed using the patch method (B). This terrain corresponds to the northwestern corner of Figure 13. This portion of the elevation map has a range of 163 meters. Notice that some smoothing has occurred.

Figure 15: A difference map that shows the error between (A) and (B) from Figure 14. The error has a range of 0..29 meters.

We also investigate the impact of using different patch sizes on both running time and reconstruction accuracy. In Table 2, we use patch sizes from $20 \times 20$ to $400 \times 400$, and record the running time, mean absolute error, maximum error, and RMS error of the reconstruction. We can see that using a patch size of $100 \times 100$ gives a nice balance between accuracy and speed. Table 3 shows the small amount of error when comparing the patch version of ODETLAP to the non-patch version. By examining the amount of error introduced versus the overall speedup that is gained, using patches that are $100 \times 100$ in size produces very good results.

The patch-based ODETLAP algorithm provides two performance improvements. The first is from decomposing large-scale terrain data into small patches and running ODETLAP sequentially on each of them, which we call serialized ODETLAP. Figure 17 shows the running time comparison of the non-patch version of ODETLAP and serialized patch-based ODETLAP. From the figure, we can see that the running time is greatly reduced even if no parallelism is used.

The second speedup occurs because we are running ODETLAP on multiple patches concurrently in a parallel environment. In Figure 18 we show the total running time for a test case with 1521 patches when run on various numbers of processors. We see that parallelism provides an excellent speedup using up to 127 worker processes. Beyond that, the overhead of parallelism becomes significant. We have opportunities to improve the performance even further. We will need to look closely at ways to optimize disk I/O, which is currently our primary bottleneck. We will discuss these improvements in more detail in Section 6.

In Table 4, we present the running time and accuracy information for all three ODETLAP versions. The size of the input terrain data is $800 \times 800$, and the mean elevation is 107. We can see that the patch method only increases errors by approximately 0.1% when compared with the non-patch method, and the running time is reduced to about 0.2% of the original.

6. FUTURE AND ONGOING WORK

In our implementation, the size of the input elevation map is limited by the the amount of memory required to store the entire grid while merging it. In order to overcome this limitation, we are working on streaming the output to the disk as it is merged and freeing the memory for completed patches. Also, the implementation has a bottleneck when reading the input data from disk. The input data set is loaded once for every patch that needs to be calculated. For example, when calculating ODETLAP on a $16000 \times 16000$ grid, there are...
Table 2: Comparison of different patch sizes: Starting with original terrain of size 2000 × 2000, input points are sampled every 10 points in the x and y directions, thereby selecting a total of 1% of the total points. A patch size of 100 × 100 gives a good compromise between running time and accuracy. Reported errors in these examples are with respect to the original terrain data set.

<table>
<thead>
<tr>
<th>Patch Size</th>
<th>Time (s)</th>
<th>Mean Error</th>
<th>Max Error</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 × 20</td>
<td>0m14s</td>
<td>0.6570</td>
<td>13</td>
<td>0.9798</td>
</tr>
<tr>
<td>40 × 40</td>
<td>0m7s</td>
<td>0.6619</td>
<td>13</td>
<td>0.9594</td>
</tr>
<tr>
<td>50 × 50</td>
<td>0m8s</td>
<td>0.6640</td>
<td>13</td>
<td>0.9617</td>
</tr>
<tr>
<td>100 × 100</td>
<td>0m9s</td>
<td>0.6598</td>
<td>13</td>
<td>0.9530</td>
</tr>
<tr>
<td>200 × 200</td>
<td>0m25s</td>
<td>0.6598</td>
<td>13</td>
<td>0.9527</td>
</tr>
<tr>
<td>400 × 400</td>
<td>1m28s</td>
<td>0.6598</td>
<td>13</td>
<td>0.9527</td>
</tr>
</tbody>
</table>

Table 3: This companion data to Table 2 reports the errors of Patch ODETLAP with respect to Non-Patch ODETLAP processing of the same test data consisting of 800 × 800 points. Note that the parallel version introduces only a very small amount of error.

<table>
<thead>
<tr>
<th>Patch Size</th>
<th>Mean Error</th>
<th>Max Error</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 × 20</td>
<td>0.1431</td>
<td>2</td>
<td>0.3801</td>
</tr>
<tr>
<td>40 × 40</td>
<td>0.0532</td>
<td>1</td>
<td>0.2307</td>
</tr>
<tr>
<td>50 × 50</td>
<td>0.0634</td>
<td>2</td>
<td>0.2519</td>
</tr>
<tr>
<td>100 × 100</td>
<td>0.0149</td>
<td>1</td>
<td>0.1223</td>
</tr>
<tr>
<td>200 × 200</td>
<td>0.0039</td>
<td>1</td>
<td>0.0628</td>
</tr>
<tr>
<td>400 × 400</td>
<td>0.0008</td>
<td>1</td>
<td>0.0291</td>
</tr>
</tbody>
</table>

101761 patches, and the file is loaded once for each of those patches. When we use more processors, it means that more processors are all trying to read the file simultaneously. This results in limited scalability. We have designed the next version so that the input only needs to be accessed by a single source process. The source process distributes the data to the worker processes, thereby alleviating the bottleneck. A further improvement would be to allocate multiple source processes for larger data sets, allowing us to run ODETLAP using any number of available processes efficiently.

The patch method described in this paper creates an artificial coupling of patch size with the amount of overlap needed for accurate results. The algorithm would gain flexibility if a patch represented an independent square of data that did not overlap with any other data patches, and the amount of overlap were represented separately. For example, one could specify a patch size of 50 × 50 with an overlap of 25 pixels in each direction. ODETLAP would then be calculated on 100 × 100 blocks except for patches that lie in the edges and corners of the data set. Abstracting the overlap this way is described as a “halo” in [6]. The patch size would be selected to optimize speed, and the amount of overlap would be selected to ensure an acceptable accuracy in the results. Implementing the patch method in this way would lend itself to a multigrid implementation and to striding the data set across multiple processors.

As shown in Figure 12, there are still some errors compared with results from the non-patch version of ODETLAP, which means that we may still be able to improve the patch merging process to get higher accuracy. We would need to analyze the results and find out what causes the errors, and choose the most appropriate interpolation scheme based on our findings. This can be combined with strategies to optimize for elevation maps with irregular sampling such as point cloud data or Triangulated Irregular Network (TIN) data. For example, patches with more points should be weighted more heavily than undersampled patches when doing interpolation in the overlapped regions. Additionally, multigrid could be used as a point selection strategy in which high frequency areas are more densely sampled than flat areas. Integrating a multigrid approach with the parallel patch method would lead to interesting research.

We have performed initial test runs on the 32,768 processor Blue Gene/L system that is part of Rensselaer’s Computational Center for Nanotechnology Innovations (CCNI). With 32,768 processors, the Blue Gene/L system will enable us to get much faster performance than what could otherwise be possible. The Blue Gene presents a challenge because each processor is limited to 512 MB of memory in its optimal configuration. For this reason, we have designed two running modes for the Blue Gene. The first mode is optimized for data sets that are small enough to fit into a single process’ memory. In this mode, the data is read from disk, distributed to the workers, and merged in the sink before being written to the disk. This gives the fastest execution time, but will not run for very large data sets (16000 × 16000 and larger). The second mode trades the speed benefits of the first method for nearly limitless data set sizes by us-
Figure 18: We reconstruct a $2000 \times 2000$ grid from a subset of 1% of the original points by running ODETLAP on a distributed platform consisting of a cluster of AMD Opterons running at 2.6GHz. We were able to achieve a linear decrease in running time for up to 128 processors before the overhead of file I/O prevents us from gaining any additional speedup without further optimizations. The algorithm includes a central process to merge results, but this process is not included in the x axis.

7. CONCLUSIONS

In this paper we have presented our recent progress in parallel terrain compression and reconstruction that processes digital elevation maps of sizes as large as $16000 \times 16000$. We use ODETLAP to reconstruct a terrain map from sparse, isolated samples. In order to greatly increase the efficiency of ODETLAP, we divide the original terrain into patches which are then run on multiple processors in parallel. A patch size of 100 $\times$ 100 provided a very good performance gain, while minimally impacting the results of output when compared to the non-patch version of ODETLAP. The results from experiments show that our method greatly reduces the running time and ensures a high quality in the reconstructed image. This new technique changes the way that we will approach large terrains in the future.

8. ACKNOWLEDGMENTS

This research was supported by NSF grants CCR-0306502 and DMS-0327634, by DARPA/IPTO/GeoStar, and by CNPq - the Brazilian Council of Technological and Scientific Development. We thank Chris Stuetzle and Metin Inanc for valuable discussions on terrain representation and compression. We also thank Professor Christopher D. Carothers for introducing us to parallel computing, and for his assistance with the parallelization of our algorithm.

9. REFERENCES

Parallel ODETLAP for Terrain Compression and Reconstruction

Jared Stookey
Zhongyi Xie
W. Randolph Franklin

Dan Tracy
Barbara Cutler
Marcus V. A. Andrade

Outline

- Quick Overview of our research
- ODETLAP (Non-Patch)
- Motivation for Parallelization
- Our Approach
- MPI Implementation
- Results
- Current and Future Work
Quick Overview

- Our research
  - Terrain compression
  - Compress terrain by selecting subset of points
  - Reconstruct the terrain by solving a system of equations to fill in missing points
  - The method we use to reconstruct the terrain is slow for large datasets
  - We came up with a method for reconstructing very large datasets quickly using MPI

ODETLAP

- Over-Determined Laplacian
- Two Equations:
  - \( 4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} \)
  - \( z_{ij} = h_{ij} \)
  - Multiple values for some points
  - Require a smooth parameter \( R \) to interpolate when multiple values exist
- Reconstruct an approximated surface from \( \{h_{ij}\} \) (Red points)
ODETLAP Compression

- Lossily compress image by selecting subset of points
- ODETLAP reconstruction solves for the whole terrain

1) Compress  2) Store

3) Reconstruct (ODETLAP)

Motivation for Parallelization

- ODETLAP prohibitively slow for large datasets
- We need a scalable implementation
- Only a small neighborhood of points will affect a particular elevation.
- 1 pixel only affected an area of 62x62
Our Approach

- Divide the terrain into individual patches
- Run ODETLAP on each patch separately

1) Compressed terrain
2) Divide it into patches
3) Reconstruct each patch
4) Merge the patches

There is a problem! (continued)

- We get discontinuity if we naively merge the patches

Naively reconstructed terrain: Errors:
There is a problem!

- Points near the edges of patches have incomplete data which causes errors

Pixels in red show erroneous results

Solution

- Use overlapping layers of patches
Solution

- Use overlapping layers of patches
Solution

- Use overlapping layers of patches
Solution

- Use overlapping layers of patches
Solution

- Use overlapping layers of patches
Solution

- Use overlapping layers of patches
- Then merge the results
Problem: Averaging the patches

- A simple averaging of the patches incorporates the border error into the reconstructed terrain:

Solution: Bilinear Interpolation

- Use bilinear interpolation to do a weighted average such that border values fall off to zero:
Weighting Pattern for Bilinear Interpolation vs. Simple Averaging

MPI Implementation

1) Each processor (except central process) is pre-assigned one or more patches
2) Every MPI process does the following for each patch assigned to it:
   - Load patch
   - Run ODETLAP on the patch
   - MPI_send the patch to the central process
3) When all of the patches have been received by the central process, merge them using bilinear interpolation.
Results

- 16,000x16,000 Central USA terrain data
- Use 128 2.6 GHz processors on RPI CCNI cluster
- Divide into 101,761 patches of 100x100 size
- Completed in 28 minutes and 32 seconds
- Non-patch ODETLAP would have taken 179 days

Results (cont.)

- Size: 16Kx16K
- STD: 217
- Range: 1013
- Mean Error: 1.96
- Max Error: 50
- RMS Error: 2.76

The terrain was compressed by a factor of 100, with a mean error within 0.2% of the range.
Original and reconstructed Terrain

Patch Size vs. Time & Error

<table>
<thead>
<tr>
<th>Total size</th>
<th>#points used</th>
<th>Patch size</th>
<th>Running time</th>
<th>Mean absolute Error</th>
<th>Max Error</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000*2000</td>
<td>39894</td>
<td>50*50</td>
<td>13m 39s</td>
<td>0.6640</td>
<td>13</td>
<td>0.9617</td>
</tr>
<tr>
<td>2000*2000</td>
<td>39894</td>
<td>100*100</td>
<td>5m 55s</td>
<td>0.6598</td>
<td>13</td>
<td>0.9530</td>
</tr>
<tr>
<td>2000*2000</td>
<td>39894</td>
<td>200*200</td>
<td>5m 25s</td>
<td>0.6598</td>
<td>13</td>
<td>0.9527</td>
</tr>
<tr>
<td>2000*2000</td>
<td>39894</td>
<td>400*400</td>
<td>18m 49s</td>
<td>0.6598</td>
<td>13</td>
<td>0.9527</td>
</tr>
</tbody>
</table>

These results come from an 8-processor machine
Serialized vs. Parallel

- Serialized: A single worker processor runs each patch sequentially (speedup of 9.5 in the test)
- Parallel: Several processors run on many patches in parallel (additional speedup of 5.6 in the test)

Test data: 800 x 800 size with mean elevation of 107

Running Time Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th>Running time</th>
<th>Mean Error</th>
<th>Max Error</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original ODETLAP</td>
<td>549s</td>
<td>0.6150</td>
<td>7</td>
<td>0.8835</td>
</tr>
<tr>
<td>Serial ODETLAP</td>
<td>34s</td>
<td>0.6156</td>
<td>7</td>
<td>0.8846</td>
</tr>
<tr>
<td>Parallel ODETLAP</td>
<td>9s</td>
<td>0.6156</td>
<td>7</td>
<td>0.8846</td>
</tr>
</tbody>
</table>

Test data: 800 x 800 size with mean elevation of 107, run on 8 processors. Parallel ODETLAP is 50 times faster, while introducing only 0.1% additional error.
Current and Future Work

- Improvements to our implementation
  - Reduce data size – regular grid can be more compact
  - Each process should grab the next available patch
  - Optimize for the Blue Gene/L system (see next slide)
- Reduce errors from the patch method
  - Improve the method for merging patches

Blue Gene/L System

- Computational Center for Nanotechnology Innovations (CCNI) at RPI
- 32,768 CPU’s @ 700 Mhz
- 512-1024MB memory/CPU (non-shared)
- Opportunity to run very large data sets quickly
- New method
  - Source, Sink, Workers, and Coordinator
  - DEM size is not limited by process memory size
  - Use processors as cache instead of the disk
    - On the BG, disk is slow, network and memory is very fast
    - We must reduce the overhead to take advantage of all CPU’s
Path Planning on a Compressed Terrain

Daniel M. Tracy
Rensselaer Polytechnic Institute
tracyd@cs.rpi.edu

W. Randolph Franklin
Rensselaer Polytechnic Institute
frankwr@rpi.edu

Barbara Cutler
Rensselaer Polytechnic Institute
cutler@cs.rpi.edu

Franklin T. Luk
Rensselaer Polytechnic Institute
luk@cs.rpi.edu

Marcus Andrade
Federal University of Viçosa, Brazil
marcus.ufv@gmail.com

ABSTRACT

We present a better algorithm for path planning on complex terrain in the presence of observers and define several metrics related to path planning to evaluate the quality of various terrain compression strategies.

The path-planning algorithm simulates a smugglers and border guards scenario. First, we place observers on a terrain so as to optimize their visible coverage area. Next, we compute a path that a smuggler would take to minimize detection by an observer, path length, and uphill movement. The smuggler is allowed the full range of Euclidean motion on the 2-dimensional plane, unlike alternate path planning schemes that strictly avoid obstacles. We use two runs of the A* algorithm to efficiently compute this path.

We introduce new application-specific error metrics for evaluating lossy terrain compression. The target terrain applications are the optimal placement of observers on a landscape and the navigation through the terrain by smugglers. The error metrics compare the observer visibility and the cost of the optimal smuggler’s route on the reconstructed terrain to the original terrain.

Categories and Subject Descriptors

I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling; E.4 [Data]: Coding and Information Theory—Data compaction and compression

Keywords

Path planning, observer viewshed, terrain compression.

1. INTRODUCTION

Recent advances in LIDAR and related scanning technologies have allowed for the acquisition of increasingly high resolution terrain data, in the form of rasters of elevation data. A sample urban dataset we are using for testing covers approximately 25 city blocks, containing 12 million 3D points and requires about 300 MB for storage. Publicly-available terrain datasets covering the continental United States, sampled horizontal approximately every 30 meters require 40 GB of storage. Higher resolution data sampled every 10 meters or less is available and becoming commonplace.

These types of data sets have many commercial and military applications. For example, a soldier may wish to download the elevation data for a war zone onto a PDA, in order to plan a safe and efficient route through the terrain. Due to limited bandwidth and storage, it is typically not practical to download and process the original data. Thus, we need to transmit and store this information in a compressed format.

Depending on the application, a certain level of information loss can be acceptable. In the field of image processing, the usual way to evaluate a lossy compression scheme is to calculate the per-pixel average and the maximum errors between the original and reconstructed geometries. The standard image quality metrics may not be appropriate for the common tasks performed on terrain data, and it may be beneficial to consider more domain-specific applications. For example, correct evaluation of inter-elevation point terrain visibility is essential for surveying, cell phone tower placement, military surveillance, etc.

We consider the problem of multiple observer siting on a compressed and reconstructed terrain. An observer is placed at a point on the terrain and can see other points on the terrain only if no other part of the landscape obstructs a direct line of sight between the points, and the point is within the radius of visibility. One application of observer siting is to maximize the amount of visible terrain by specifying the optimal placement for a fixed number of watchtowers. In many real world scenarios, this problem must be solved using only the reconstructed terrain because the original terrain is prohibitive for transmission and storage.

Another interesting terrain data application is that of path planning. Given a complex mountainous or urban dataset, determine the shortest path through the terrain minimizing various relevant quantities such as total path length, distance traveled through terrain visible by one or more observers, and distance traveled uphill. We present a method for efficiently computing the best path through a terrain and show how this target application can be used as a metric for evaluating terrain compression techniques.

The contributions of our paper are:

• A new application-specific method for evaluating the
quality of the reconstructed geometry provided by a particular compression scheme.

- A novel scheme for computing a minimum length path between two terrain points while avoiding detection.
- New compression error metrics based on path length and visibility errors.

2. BACKGROUND

2.1 Path Planning Techniques

Path planning around obstacles in \( E^2 \) and \( E^3 \) is a classic research topic with diverse important applications. The well-known Lee maze-running algorithm [9] and High-tower line-search algorithm [5] consider movement in only four directions on a two-dimensional terrain.

In robotics, the goal is often to find the safest path, keeping as far away from obstacles as possible. To accomplish this, a Voronoi diagram of the obstacle boundaries may be constructed. Performing a graph search on the Voronoi diagram can then yield the optimal path [1]. Another approach is to construct the visibility graph for a two-dimensional terrain from the vertices of polygonal obstacles. Performing a graph search on the visibility graph will yield the shortest path. It is not necessary to restrict the obstacles or the path to a grid [10]. This will work well if simply the Euclidean distance is to be minimized. However, for more complex cost functions, a different approach is needed.

Three-dimensional path-planning problems are much more difficult. The general case of computing the shortest Euclidean path between two points, while avoiding polyhedral obstacles, is NP-hard. A common problem in robotics is to compute a collision-free path of a 3-dimensional object, such as a 6-DOF human arm, moving through a 3-dimensional workspace [8]. Kavraki [7] computes collision-free paths for robots moving among stationary obstacles. This method is particularly interesting for robots with many degrees of freedom (five or more).

The A* algorithm is a general-purpose graph search algorithm which can be adapted to solve any path-planning problem whose search space can be expressed as a tree [4]. This serves as the base for our path planning algorithm, described later in the paper.

2.2 Multiple Observer Siting

The goal in the placement of observers in the “smugglers and border guards scenario” is to maximize their joint viewed, which is the area of the terrain that is visible by at least one observer. Due to the inherent complexity of computing the visibility between every pair of points on the terrain, it is impractical to compute the exact optimal solution to the multiple observer siting problem, especially when considering applications on small portable devices used by soldiers out in the field. Instead the randomized multiple observer siting algorithm by Franklin and Vogt [3] is used to approximate the optimal siting of observers.

3. OUR CONTRIBUTIONS

In this section we describe our efficient algorithm for planning smuggler’s paths through complex terrain in the presence of observers. We then describe a set of metrics for evaluating how well the terrain details are preserved with different lossy compression schemes.
second pass is more efficient than the first pass. Examples of both passes are illustrated in Figure 2.

### 3.3 Alternate Cost Functions

We also investigated additional extensions to the cost metric. First we added a penalty for traveling uphill. Next, we lifted the restriction on traveling through regions that were visible from one or more of the observer positions to allow this movement but carry a stiff penalty. The cost of moving from one point to an adjacent point is given by:

$$\text{Cost} = \sqrt{h^2 + v^2} + S \times V,$$

where $h$ is the horizontal distance between the points, and $v$ is the slope penalty. $S$ is the slope penalty, which is $(1 + v/h)$ when going uphill and 1 otherwise. $V$ is the visibility penalty, which is 100 if the cell is visible and 1 otherwise. A penalty of 100 was found to be appropriate to discourage movement through a viewed area when it is avoidable.

The cost metric parameters may be adapted to different types of terrain scenarios. For example, for urban LIDAR data, a visibility penalty of 10 and a slope penalty of $1 + v^2/h$ works better in discouraging the smuggler to climb up the side of a building.

Calculated cost to move between adjacent points is trivial. However, for the two-pass system that allows a full range of Euclidean motion, the cost to traverse a straight line that connects two distant points must be computed. This line is not likely to pass through grid points exactly. Here the elevation and visibility at several points along the line must be interpolated.

Our path planning procedure takes a cost function defined on a uniform grid, with non-uniform edge weights, and computes the path that minimizes the cost function while allowing a full range of Euclidean motion.

### 3.4 Novel Evaluation Metrics

We have innovated a new application-specific protocol for evaluating the quality of terrain compression. First, the original terrain representation is compressed then reconstructed to obtain the alternate representation. This is the representation available to field agents who will be making tactical decisions. Second, the multiple observer siting is performed on the alternate representation to produce a set of observers, and their joint viewshed is computed. Third, this same set of observer positions is transferred back to the original representation, where their true joint viewshed is computed. This simulates the errors and non-optimality that would result if observer placement was computed in the field using a compressed representation. In the fourth step, our path planning algorithm is applied to find the optimal paths on both the original and the reconstructed representations.

Three new error metrics were derived for the smugglers and border guards scenario. They target typical applications and tasks that use terrain data. If artifacts from a proposed compression scheme lead to significant errors in any of these metrics, the compression scheme should not be recommended for critical terrain applications, even if the reconstructed terrain is without visual artifacts.

The new error metrics are as follows:

1. **Viewshed Error**, or area of the symmetric difference of the cumulative viewsheds, implying suboptimal observer siting.
2. **Path Cost Error**, or difference in the path costs computed on the original and alternate, implying that the smuggler’s path is suboptimal.
3. **Path Visibility Error**, or percent of the alternate path that is visible when it is applied back to the original terrain. The smuggler plans the path using the alternate representation, believing that path to be safe, but which might inadvertently traverse a guard’s field of vision.

### 4. RESULTS

The new protocol and error metrics were tested on two compression schemes: JPEG 2000 [6] [11] and ODETLAP (Overdetermined Laplacian Solver) [2]. 400 × 400 terrains sampled from DTED-2 data were used because both JPEG and ODETLAP can process them efficiently, though the path planning itself has been observed to be efficient on 3200 × 3200 terrains. Three hilly (hill1, hill2, hill3) and three mountainous datasets (mtn1, mtn2, mtn3) were chosen to standardize all testing. Sample paths on each are shown in Figure 3. Some initial experiments have also been performed on some urban LIDAR data from Ottawa, as shown in Figure 4.

The scheme is robust, working equally well on JPEG 2000 and ODETLAP. The numbers agree with what is seen visually. The greater the compression, the more the terrain features are blurred, and the larger the computed errors. The visibility is usually greater on the alternate representation than on the original. This is because the compression removes detail and smoothes out the terrain, eliminating visibility obstructions. The increased visibility will sometimes block out important passages, forcing the smuggler to take a long detour. The path visibility errors tend to be very small because a portion of the terrain that is biased towards the nonvisible areas is being sampled. This is a good indication that we are computing correct paths.

Also, our heuristic path planning algorithm was compared against the brute-force method. The average difference in the lengths of the computed paths was less than 0.1%, while the average speedup was greater than 100.

### 5. CONCLUSIONS

Observers were optimally sited on a terrain and good smuggler’s paths were computed through the terrain. An algorithm was developed that minimizes path cost, as measured by path length plus penalties for uphill movement and observer detection, while also permitting the range of Euclidean motion, rather than simply minimizing Chebyshev distance. New methods were also developed for evaluating terrain compression. These application-specific error metrics test how well the reconstructed terrain performs on the smugglers and border guards problem, with regards to the observers’ visibility and the smuggler’s path cost. When a user wants to plan a smugglers and border guards scenario on a reduced representation of a terrain, these new error metrics will help select the appropriate representation.

### 6. FUTURE WORK

Alternative observer placement policies may also be considered. For example, rather than seeking to maximize the total coverage area, the observers could form a perimeter and seek to minimize the “gaps” in that perimeter.

One useful extension of the smugglers and border guards scenario is to consider mobile observers. Each guard patrols a specified path. The smuggler may have to pause at certain
intervals to wait for the guards to reposition themselves. An element of unpredictability may also be added to the guards’ movements. The smuggler will not know the observers’ future positions, though it would retain the ability to track the observers’ current positions.

7. ACKNOWLEDGEMENTS

This research was supported by NSF grants CCR-0306502 and DMS-0327634, by DARPA/DSO/GeoStar, and by CNPq - the Brazilian Council of Technological and Scientific Development.

8. ADDITIONAL AUTHORS

Additional authors: Metin Inanc (Rensselaer Polytechnic Institute, email: inancm@gmail.com), Zhongyi Xie (Rensselaer Polytechnic Institute, email: xie@cs.rpi.edu), and Jared Stookey (Rensselaer Polytechnic Institute, email: jstookey@jstookey.com).

9. REFERENCES


Path Planning on a Compressed Terrain

Daniel M. Tracy, W. Randolph Franklin, Barbara Cutler, Marcus Andrade, Frank Luk, Metin Inanc, Zhongyi Xie, Jared Stookey
Rensselaer Polytechnic Institute

We simulate a smugglers and border guards scenario to evaluate terrain compression. Multiple Observer Siting is used to place observers so that they jointly see as much of the terrain as possible. Our path planning algorithm computes a smuggler’s path.

Compress the terrain then uncompress it to generate the alternate representation.

Site the same group of observers on the original terrain, and compute the new joint viewsched.

Evaluate how well visibility is preserved, and compute smugglers’ paths to evaluate how well path costs are preserved.

The smuggler’s cost metric includes penalties for observer detection and uphill movement: not simply distance around obstacles.

Our two pass path planner efficiently allows a full range of Euclidean motion on a raster terrain.

Compute many separate paths to sample a larger portion of the terrain.

Our path planning algorithm:
1) Accounts for complex cost metrics
2) Allows full range of Euclidean motion on a 2D grid
3) Is efficient on high-res terrain

Red: High elevation
Blue: Low elevation
Bright: Visible
Dark: Hidden

A sample smuggler’s path on a 400x400 terrain.

Perform multiple-observer siting on the alternate representation to generate a set observers, along with the corresponding joint viewsched

1st pass: Chebyshev
2nd pass: Euclidean

Smuggler’s path on 2000x2000 terrain derived from Ottawa LIDAR scan
Path Planning on a Compressed Terrain

Daniel M. Tracy, W. Randolph Franklin, Barbara Cutler, Franklin T. Luk, Marcus Andrade, Metin Inanc, Zhongyi Xie, Jared Stookey

Rensselaer Polytechnic Institute

---

Path Planning

- Smuggler’s Path: Find the shortest path between two given points while trying to avoid detection by the observers.
- A* algorithm
- Add penalty for going uphill.
Cost Metric

- Cost of moving from one cell to an adjacent cell:
  \[ Cost = \sqrt{(h^2 + v^2)} \times \text{SlopePenalty} \times \text{VisibilityPenalty} \]

- \( h \) is the horizontal distance.
- \( v \) is the elevation difference.
- \( \text{SlopePenalty} \) is \( 1 + \frac{v}{h} \) when going uphill and 1 otherwise.
- \( \text{VisibilityPenalty} \) is 1 if the new cell is not visible and 100 otherwise.

Range of Motion

Chebyshev  Euclidean

A straightforward application of the A* algorithm results in the Chebyshev distance being minimized, rather than the Euclidean distance.
Path Planning

- New approach: Two pass system
- First pass: Plan a path that minimizes Chebyshev distance.
- Second pass: Only include points from the first path in the search space.
- Not guaranteed to be optimal, but in practice it often is.
Evaluating Hydrology Preservation of Simplified Terrain Representations

Jonathan Muckel, Christopher Stuetzler, W. Randolph Franklin, Marcus Andrade, Barbara Cutler, Melin Imano, Zhongyi Xie
Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute
University Federal de Vitoria, Vitoria Brazil
Department of Computer Science, Rensselaer Polytechnic Institute

PROBLEM
- Large Dataset Sizes: Terrain data is being sampled at ever increasing resolutions over larger geographic areas requiring special compression techniques to manage the data.
- Simulating Issues: Dataset inaccuracies due to insignificant resolution sampling and data collection issues impedes water flow causing small and unrealistic water features.
- Measuring Effectiveness: Typically, terrain compression algorithms focus on minimizing L2-norm squared errors and maximum error. These metrics fail to capture whether a reconstructed terrain preserves the drainage network.

MEASURING HYDROLOGY ERROR

To determine the amount of hydrology error lost during terrain simplification the drainage network computed on the reconstructed geometry is mapped onto the original elevation matrix.

\[
\text{Energy}_{\text{Error}} = \sum_{ij} \max(0,E_{ij} - E_{ij(i)}) \times W_{ij}
\]

COMPRESSING HYDROLOGY FEATURES

The flow chart above describes the methodology behind the ridge-ridge compression method. Input are in boxes and programs in circles.
1. We compute the drainage network and ridge network. This locates the valleys and ridges.
2. Next, we eliminate redundancy in the network by taking the most significant points using the Douglas-Pageau method.
3. We use DCELAP to interpolate the surface based on a subset of known, hydrology significant points.

RESULTS

<table>
<thead>
<tr>
<th>Compute</th>
<th>Ridge-Dense</th>
<th>JPED-2000</th>
<th>VSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.0401</td>
<td>0.1340</td>
<td>0.1459</td>
</tr>
<tr>
<td>0.2</td>
<td>0.0201</td>
<td>0.0400</td>
<td>0.0100</td>
</tr>
<tr>
<td>0.3</td>
<td>0.0001</td>
<td>0.0100</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.4</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Each dataset is compressed by 3 different lossy compression schemes at 3 different levels. For each, the percent of low upland and the energy error metric is presented.

To reduce error in our compression technique, we add k number of points with the highest error.

Points that are within a certain distance of each newly added points are forbidden from being added.

RPI Geo* Final Report 299/919 ACMGIS 2008 poster 3
Evaluating Hydrology Preservation of Simplified Terrain Representations
Jonathan Muckell and Christopher Stuetzle
Advisors: Marcus Andrade, Randolph Franklin, Barb Cutler
Rensselaer Polytechnic Institute

Measuring Hydrology Error
Measuring Hydrology Error

- Error is reduced for flow traveling downhill, increased for flow traveling uphill.
- Error is determined by gradient and amount of flow.

$$\text{Energy}_{\text{Down}} = \sum \max(0, E_i - E_{i+1}) \cdot W_i$$

$$\text{Energy}_{\text{Up}} = \sum \max(0, E_{i-1} - E_i) \cdot W_i$$

$$\text{Error} = \frac{\text{Energy}_{\text{Up}}}{\text{Energy}_{\text{Down}}}$$
4 Program Review Presentations and Reports

Kickoff talk — Mar 2005

Kickoff talk transcript — Mar 2005

Site visit — Jun 2005

Progress report — Jul 2005

Progress report — Aug 2005

Progress report — Sep 2005

TINning very large datasets — Oct 2005

Geologically correct and compact terrain representation — Nov 2005 review

Nov 2005 status report

Geologically correct and compact terrain representation — Jan 2006 status report

Geologically correct and compact terrain representation — Feb 2006 site visit

Compact visibility-preserving terrain representations — Apr 2006

Progress report — May 2006

Summary slide — May 2006

Annual report — Jun 2006

Compact visibility and path preserving terrain representations — Aug 2006

Geo* Phase I achievements — Nov 2006

Compact visibility and path preserving terrain representations — Apr 2007 review

RPI Geo* Phase II

Summary for NGA — Aug 2007

Oct 2007 review: DEM compression and terrain approximation — smugglers and border guards

Nov 2007 Slope compression retasking

Feb 2008 Task Summary

Geo* at RPI — Jun 2008 site visit
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hydrology aware triangulation of terrain data – Jun 2008 site visit poster</td>
<td>670</td>
</tr>
<tr>
<td>Hydrology aware triangulation of terrain data – Jun 2008 site visit talk</td>
<td>671</td>
</tr>
<tr>
<td>Path Planning and Slope Representation of a Compressed Terrain – Jun 2008 site visit</td>
<td>684</td>
</tr>
<tr>
<td>Aug 2008 review: Geo* at RPI</td>
<td>706</td>
</tr>
</tbody>
</table>
Geologically Correct Terrain Representations & Radar Siting

WR Franklin
Rensselaer Polytechnic Institute
wrf@ecse.rpi.edu, 703-447-7808

DARPA/DSO/Geo* Kickoff meeting
Savannah, March 29 2005

Team

• W Randolph Franklin, Rensselaer Polytechnic Institute, Center for Subsurface Sensing and Imaging Systems – leader, research
• COL Clark K Ray, USMA – advising
• Duane Niemeyer, ESRI – develop ArcGIS applications and transfer technology into NGA programs
Goal

• Represent terrain (terrestrial elevation)
• Compactly; lossily
• Bias representations towards legal terrain
• Preserving usefulness for operations
  – Multiple observer siting with intervisibility
• Implement on the largest datasets.
• Study size / accuracy tradeoffs.
• Uncompression speed is more important than compression speed.

Deliverable

Terrain representation and operation toolkit, incorporating:
• Feature-based morphological representation,
• Overdetermined PDE representation
• Extended TIN representation
• Multiobserver siting with intervisibility operation
• Drainage operation
Impact on Defense

- More and higher-res terrain available.
- Processing power not keeping up.
- Inadequacies of current representations hinder military operations.

Lessons Learned During Afghanistan Deployment

"Next to go is your ability as a company or platoon to move from point A to B without injuries, or people falling behind. Once you get in the mountains that 5 clicks you started out on turns into 9-11 clicks because of all the deep ravines and mountains that didn't make it on the map because of the contour interval."

"Plugger battle drill is the way to go, even with the civilian models; the contour interval on the maps is outrageous so terrain association was difficult."

"Landing was the most dangerous part while we were there just because of the conditions and terrain.

- 1SG Romero
Project Outline

• Representations
  – Feature-based operators like scooping
  – Overdetermined PDEs
  – Extensions to TINs

• Operations
  – Multiobserver siting with intervisibility
  – Drainage networks
First: Study Terrain Properties

- Many local max, few local min
- Long range order - rivers
- Elevation and slope discontinuities are common, and are very important for mobility and visibility

Which is land, which water?

You can answer this => there is unexploited structure.
Where is This?

Answer
Fourier Series

- Widely used
- Excellent for representing many physical phenomena, like vibrations.
- Quite unsuitable for terrain.
- They assume $C^\infty$ continuity
- The truncated series is too smooth
- ...and has many local minima

Project Outline

- Representations
  - Feature-based operators like scooping
  - Overdetermined PDEs
  - Extensions to TINs

- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks
Feature-Based Operators

- Use morphological operators that, to the extent reasonable, represent the physics of terrain generation.
- Initially: represent water erosion.
- Implement as a **scooping** operator.

Properties

- Feature-based operators are not linearly superimposable.
- Therefore computation will be compute-intensive.
- But reconstruction will be fast.
- Truncating a series produces a valid, tho less accurate terrain.
Simulation

Details

- Incremental progress, based on experiments.
- First, refine scoops.
- Scoop along hand-drawn, then computed flowlines.
- Process the residual surface, and scoop it.
- Then, add other operators, like *plop* and *scrape.*
Project Outline

- Representations
  - Feature-based operators like scooping
    - Overdetermined PDEs
  - Extensions to TINs

- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks

Overdetermined PDEs

- Purpose:
  - Interpolate from contour and point data to an elevation matrix
  - Do it better than previous methods.

- Impact:
  - Make large amounts of existing data more easily available at higher quality
**Particular Advantages**

- Can process both whole and incomplete contours.
- Can process isolated points.
- Infers local maxima inside topmost contours.
- Introduces no terracing artifacts on contours.
- Kidney shaped contours are no problem.

**Method**

- Set up a linear system whose unknowns are all the elevations, known and unknowns.
- Two types of equations:
  - All points: \( z_{ij} = (z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1})/4 \)
  - Known points: \( z_{ij} = h_{ij} \)
  - (Each known point has 2 equations.)
  - Relative weights trade off accuracy and smoothness.
- Solve this overdetermined system.
Status

- John Childs masters project did a first version.
- Future:
  - Thorough evaluation of effect on siting of different accuracies on many datasets.
  - Extend testbed to larger datasets.
  - Productize.
Project Outline

- Representations
  - Feature-based operators like scooping
  - Overdetermined PDEs
  - **Extensions to TINs**

- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks

Extensions to TINs

- Handle breaklines, discontinuities automatically w/o user assistance.
- Handle large datasets
- Quickly
- Splines (their effectiveness depends on C^2 continuity, contracting earlier hypothesis
- Well…. This is research!
Example

Original Terrain

Detail from TIN

Project Outline

• **Representations**
  – Feature-based operators like scooping
  – Overdetermined PDEs
  – Extensions to TINs

• **Operations**
  – Multiobserver siting with intervisibility
  – Drainage networks
The Operation Defines “Accuracy”

- Initially: RMS error
- Better: measure effect on operation, such as siting
- How much better is the solution computed using the best data than the solution using the more compact data?
- Formalizing this is tricky.

Project Outline

- Representations
  - Feature-based operators like scooping
  - Overdetermined PDEs
  - Extensions to TINs

- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks
Comparison Details

- Compute the operation, e.g., observer sitting, on the original data.
- Measure A = area of joint viewshed.
- Site observers on the compressed data.
- Place those observers back on original data, and compute B = their joint viewshed.
- Quality of representation = B/A.

Details - 2

- A completely different observer sitting is ok if it is just as good.
- Joint viewshed area, measured on the compressed data, can be significantly larger than when measured on the original data.
- ... proving that computing viewsheds should use the best data.
The Known Unknowns of Viewsheds

- Small changes in LOS interpolation cause large changes in visibility.
- One half of this cell has uncertain visibility.

Multiobserver Siting Steps

- Find approximate visibility index of every point in cell, using Monte Carlo sampling.
- Partition the cell into blocks and pick the best potential observers in each block.
- Using a greedy algorithm, select the best of the best observers.
- We have considerably studied the tradeoffs here.
Enforcing Intervisibility

- After the first best-of-the-best, add only new observers that are inside the joint viewshed of the previous best-of-the-best.

Effect of Intervisibility

- This reduces the joint viewshed considerably.
Reduced Resolution Effect on Siting

- Lowering horizontal resolution lowers observer sitting quality.
- Lowering vertical resolution does not as much.
- Visibility, computed on lower resolution, is too high.

Experiments

Reducing horizontal resolution

Reducing vertical resolution
Project Outline

- Representations
  - Feature-based operators like scooping
  - Overdetermined PDEs
  - Extensions to TINs

- Operations
  - Multiobserver sitting with intervisibility
  - Drainage networks

The Problem

- Computing hi-res drainage networks is slow.
- Partly caused by fake local minima that trap the water.
Solution

- We have developed a very fast connected component program.
- Use it to identify the separate fictitious basins and amalgamate them.
- Then solve a linear system for the water flow.

Impact

- Better prediction of flooding and river widths, which affects mobility.
- Tests quality of our previous terrain representations.
Project Outline

- Representations
  - Feature-based operators like scooping
  - Overdetermined PDEs
  - Extensions to TINs

- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks

Themes

1. Nonlinearity is powerful.
2. Large memory is now available.
4. … But not always.
5. Respect the terrain.
6. Find fast heuristics.
7. Discover what we certainly know.
Hypothetical Timetable

Round 1:
1. Researching feature-based morphological operators,
2. Detailing our drainage ideas,
3. Detailing our TIN ideas, and
4. Productizing our siting toolkit.

Round 2:
1. Researching feature-based morphological operators,
2. Detailing our drainage ideas,
3. Detailing our TIN ideas, and
4. Productizing our siting toolkit.

Round 3:
1. Productizing our feature-based morphological operators toolkit, and
2. Tying the pieces together into a unified terrain representation and operations toolkit.

Feedback?

• End users: please comment.
Geo* Research Plan

This report details our research plan, and complements our white paper and proposal and the powerpoint slides presented at the Savannah kickoff meeting. There are options; we await feedback on which to pursue.

1 Goals

2 Relevance to DARPA:

3 Representations:
   3.1 Feature-Based Morphological Operators Such as Scooping...
   3.2 Modelling Terrain as an Overdetermined PDE
   3.3 Enhanced Triangulated Irregular Networks

4 Operations:
   4.1 Multi-observer Siting with Intervisibility
   4.2 Efficient drainage net determination on large datasets

5 Themes
   5.1 Nonlinearity is Powerful
   5.2 Large Memory is Now Available
   5.3 Strive for Efficiency
   5.4 … But Not Always
   5.5 Respect the Terrain
   5.6 Find Fast heuristics
   5.7 Discover What We Certainly Know
1 Goals

1. The prime goal is to research and develop new representations of terrain, i.e., terrestrial elevation.

2. These representations will be lossy. That is justified by the imprecision of the original data and by the enormous savings that lossy representations allow, compared to lossless representations.

3. One facet of this is to develop representations biased towards legal terrain. That will make them more efficient.

4. Another facet is to test our results on the largest available datasets. We are uninterested in “toy” methods that do not scale up.

5. Since compression, or representation of terrain in our data structure will be performed less often than uncompression or extraction, and since compression will probably be performed at home and uncompression in the field, uncompression speed is more important than compression speed.

6. Our performance metric will be the suitability of our representations for important operations.

7. We will also, in parallel, develop better algorithms and realizations for some operations, beginning with multiple observer siting with intervisibility.

8. The induced performance metric here will be the quality of the observers sited on the compressed terrain compared to the quality of observers sited on the original terrain.

9. We will also study size / accuracy tradeoffs to make recommendations.

10. The strategic goal is a terrain representation that enables the operations, while being 100 times more compact than existing representations.

2 Relevance to DARPA:
This project contributes to the following DARPA goals:

1. Urban area operations,
2. Tracking elusive surface targets, and
3. Detection of underground structures.

## 3 Representations:

### 3.1 Feature-Based Morphological Operators Such as Scooping

#### 3.1.1 Purpose:

This is a new paradigm for terrain representation. These operators will more closely align with the properties of terrain and of how it is formed. As an analogy, Fourier series are good for representing sound waves because their math matches the physics, e.g., with linear superimposability. Terrain properties that we wish to model include these:

1. Elevation discontinuities (cliffs). These are the most important terrain properties for mobility, but are difficult to represent with traditional representations.
2. Long-range, monotonically decreasing features (rivers).
3. Lack of symmetry between up and down. There are many more local maxima than minima.

A good representation should have the progressive transmission property. Considering a particular terrain as the result of a sequence of operations, truncating the sequence should produce something that still looks like reasonable terrain, albeit only an approximation to the actual terrain. A Fourier series badly fails this test.

#### 3.1.2 Summary:

Form the terrain with a sequence of operators that add or remove material. The initial operator will be a scoop, operating as follows.

1. The initial terrain is a block.
2. A scoop is a 3D object, probably with circular vertical symmetry, such as a drill bit.
3. A scoop follows a trajectory, from some start point to the edge of the terrain. This trajectory decreases monotonically in z.

4. Unlike the summation of multiples of basis terms for a Fourier series, overlapping scoops do not sum.

5. Successive scoop operations carve out the desired terrain.

3.1.3 Parameters:

Many design choices are possible, including these:

1. How many differently shaped scoops are allowed, such as spherical and conical. Scooping is best suited for eroded plateaus.

2. Each shape’s dimensions, such as radius or angle.

3. What type of curve the trajectory is, such as a cubic spline.

The goal is to minimize the space needed to describe the terrain.

3.1.4 Details:

1. Pick some sample terrain.

2. Identify flow lines on it, initially by hand, but ultimately with our drainage program.

3. Create a testbed, probably in Matlab, to scoop along the flow lines.

4. Image the residual error, and run a second series of scoops.

5. Repeat.

6. Extend to other terrain features, such as:
   a. mountains, to be modeled by a morphological operator such as a plop,
   b. plains, to be modeled by a scrape.

7. Test on real data, making serendipitous observations, and modify appropriately.

The goal of this process is to determine the behavior of actual terrain, and then to incorporate it into the model, so that on-the-fly hand tuning in the field is not needed.

3.2 Modelling Terrain as an Overdetermined PDE
3.2.1 Purpose:

This is an alternative research idea for modeling terrain. Simultaneously pursuing competing methods incentivizes both methods to perform better, and increases the chance of one working well.

3.2.2 Problem Statement:

The universe is a cell of terrain elevations, \( z_i \). Some of the elevations are known, but most are not. We wish to approximate the unknown elevations, given the known ones. Unlike many other methods, this method handles the following desirable properties:

1. The ability to handle a wide range of inputs, including
   a. Complete contour lines
   b. Broken contours
   c. Isolated points

2. Inferring of local maxima (mountain tops) inside the innermost contours.

3. Invisibility of the contour lines in the resulting surface.

3.2.3 Impact:

This will facilitate conversion, and therefore utilization, of the large quantities of existing terrain data.

3.2.4 Technique:

Create and solve an overdetermined set of linear equations for the elevations.

1. Each post, \( z_i \), known or unknown, is a variable.

2. Each post, known or unknown, induces an equation defining it as the average of its 4 neighbors. (Border posts are slightly different.)

3. Each known post induces an equation making it equal to its known elevation. Therefore known each known post has 2 equations.

Then we solve for the elevations, using our favorite sparse system solver. E.g., Matlab can handle terrain grids up to a few hundred square, meaning perhaps 100K variables.

3.2.5 Parameters:

In an overdetermined system, each equation can be assigned a weight. Simplifying, we can weight the “averaging” equations differently from the “known height” equations. If the averaging equations are weighted more, then the surface is smoother but less accurate. Experimentally, a little inaccuracy allows a considerable smoothness.
3.2.6 Results to date:

John Childs did a masters project for Franklin on this. Small examples work nicely. Mountain tops are inferred, and modest approximations allow there to be no visible terracing in the generated surface.

3.2.7 Research:

1. Test this idea on more datasets and types of data (broken contour lines etc).

2. Implement it on larger datasets. This requires investigating sparse overdetermined matrix solvers.

3. Use more sophisticated evaluation metrics.
   a. The current metric is RMS error.
   b. Additionally, measure the quality of some operation computed on the surface. Our initial operation will be *multiobserver siting with intervisibility*.

4. Investigate the feasibility of this idea for lossily compressing surfaces as follows.
   a. Start with a complete terrain, with all posting elevations known.
   b. Select a subsample of those elevations. Various strategies are possible, such as evenly spaced points, or distinguished (in some sense) points.
   c. Compute a complete surface from that sample.
   d. Evaluate the surface.
   e. Study space/accuracy tradeoffs. "Space" refers to the space needed to store the samples, compressed.

3.3 Enhanced Triangulated Irregular Networks

The TIN is a venerable terrain representation technique, first implemented in the GIS context by Franklin, in 1973. It is still widely used, so optimizations and extensions will have an impact. Here are some possible research topics.

1. Starting from the piecewise planar TIN, compute curved patches, to test whether they represent the terrain more accurately. This works with only the triangles, and does not incorporate any new information about the original points. Therefore this method’s success depends on there being, on the average $C^1$ continuity (that is, continuity of slope) in the terrain.

2. Compute a triangular spline fit to the original points. This is more powerful than the previous point, but more complicated.
3. Evaluate the TIN on multiobserver siting quality, not just on RMS elevation error.

4 Operations:

4.1 Multi-observer Siting with Intervisibility

This, our favorite terrain operation, goes as follows.

1. Take a cell of terrain elevation postings, and

2. Some parameters, such as observer and target height, and radius of interest. An observer can see targets only with the radius of interest.

3. Site a set of observers so as to cover as much as possible of the terrain. This means to determine the locations of a quasi-minimal set of observers, the union of whose viewsheds is at least, say 90% of the terrain cell. Alternatively, fix the number of observers and then compute locations to maximize their joint coverage.

4. In addition, require that, if we define a “visibility edge” between each pair of mutually visible observers, then this graph shall be connected (the intervisibility property).

This operation has an extremely broad military impact, including:

1. Radar. sniper surveillance, and other observer siting,

2. Radio and other transmitter siting, and

3. Computation of trajectories that avoid the other side’s observers.

4.2 Efficient drainage net determination on large datasets

4.2.1 Problems:

1. Large datasets tend to require external storage algorithms, which are quite slow.

2. Many local minima occur, caused by either data errors, or by the finite sampling of a grid of elevation posting. (That is, there may be an exit channel that runs entirely between adjacent postings, and so is not captured.) These local minima trap so much of the water that long drainage patterns do not form. Hence, they must be removed.
4.2.2 Impact:

The extrinsic operation is flood prediction and control, which is important to military mobility. The intrinsic operation is as a metric to test our lossy terrain representation techniques, as follows.

1. Compute the drainage network for the terrain cell, using the best available elevation data.

2. Lossily compress the terrain using scoops, overdetermined PDEs, or whatever.

3. Compute the drainage network of the lossily compressed terrain.

4. Compare the two networks, on some metric. One might be the locations of points with water flow greater than some threshold, i.e., points that would be flooded.

4.2.3 Brief Solution:

The solution goes as follows.

1. Partition the data in separate catchment basins as follows. Considering the posts as vertices of a graph with directed edges for water flow, find the connected components. This will use CONNECT, our very fast Euclidean connected component program. Designed for 3D datasets, it handles 2D data as a special case. E.g., processing a binarized 18573x19110 image took 25 CPU seconds on a 1600 MHz laptop. Since the algorithm is linear, much larger datasets are feasible.

2. Fill in small basins, according to some policy.

3. Set up, and solve, a sparse system of linear equations for the water flow.

4.2.4 Details:

1. From Z, an NxN grid of elevation postings, create V, a 2Nx2N bit matrix.

2. Even entries correspond to elevation posts, set $V_{2i,2j}=1$ for all i,j.

3. Odd entries correspond to water flows between adjacent posts, so, for (p,q) in {(0,1), (1,0)}, set $V_{2i+1,2j+1}=1$ if water flows from $V_{i,j+1}$ to $V_{i+1,j}$. (This flow would occur when $V_j$ is the lowest neighbor of $V_{i+1,j}$. Ties must be broken carefully to prevent an infinite loop. We suggest using Simulation of Simplicity.)

4. More sophisticated methods of creating V from Z are possible if necessary.

5. Find connected components, which are the separate watersheds. We will see too many watersheds, many ending at single posts that are lower than all 4 neighbors. That is, if that one post were higher, that whole watershed would disappear.
6. Formulate a policy about which watersheds are artifacts and so should be amalgamated with neighboring watersheds. The easiest policy is to dislike any watershed whose area is less than a threshold.

7. Formulate a policy about how to remove a watershed by amalgamation. The easiest way is to raise the elevation of each post in a watershed by a large enough amount to make the watershed “convex” in a vertical sense, and higher than its neighbors. Therefore, any rain falling on this watershed will immediately flow off to a neighbor.

8. In principle, after each watershed is amalgamated, recompute watersheds (connected components) and perhaps pick another to amalgamate. In practice, this step would be optimized. However, care must be taken, since watersheds often nest. Also, erroneous nonphysical flow loops might be created.

9. It may be objected that this is too simplistic, but we believe it worth a try. More sophisticated methods are possible, which approximate filling in the depression until water overflows over the lowest border of the watershed. Really doing this seems unnecessarily slow, which is why we propose other techniques.

5 Themes

During the exploration of the technical ideas mentioned above, while exploiting serendipitous observations, certain metathemes will be followed.

5.1 Nonlinearity is Powerful.

Nonlinearity is a very powerful, albeit hard to use, approximation technique. Even for $C^\infty$, quickly convergent, functions like $\exp(x)$, the best rational approximation is more efficient than the best polynomial one. (Note that the Taylor expansion is far from the best polynomial approximation; a Chebyshev is almost optimal.) However the true power is revealed when approximating functions like $\text{abs}(x)$ or a step (Heaviside) function. Because they are $C^0$ and $C^1$, respectively, at the origin, uniform polynomial approximations do not exist. Note that the best rational approximation is more than just a Padé approximation, which is properly defined as simply a formal tranformation from a polynomial, ignoring issues of convergence.

5.2 Large Memory is Now Available.

The new 64-bit processors and operating systems, combined with cheaper memory, raise the possibility of new algorithm and data structure techniques. All the level-1 DEMs for the USA can be simultaneously stored in main memory, with room remaining for processing. The newly possible efficient random access is a powerful tool. Contrary to intuition, large
resources such as this make time and space efficiency more important, not less, since we are farther out on the asymptotic time curve.

5.3 Strive for Efficiency.

Speed of execution on large datasets is important for many reasons.

1. Many prototypes that run on toy datasets do not scale up.

2. A sufficient quantitative increase in speed causes a qualitative improvement in what’s possible.

3. Battery limitations in portable devices continue to limit the available computation power.

Time and space efficiency feedback positively on each other. E.g., I/O is faster on more compact datasets.

5.4 ... But Not Always.

Nevertheless, at certain times, efficiency is irrelevant, so long as the result is computable. For example, it is reasonable to spend serious resources maximally to compress some terrain, which will be used in the field, if decompressing is fast.

5.5 Respect the Terrain.

The challenge is that our new mathematical representation of terrain must acknowledge its properties, such as the following.

1. Real terrain is more irregular than databases such as DEM-1 cells. Algorithms tested only on that data might unknowingly be exploiting that artificial smoothness.

2. It is not differentiable many times, i.e., is generally not \( C^n \) for \( n>1 \). Indeed, the physical phenomena that generate terrain generally do not depend on, or generate, high order continuity. The major exception is the curvature, in the horizontal plane, of stream beds.

3. In places, the terrain is \( C^{-1} \), i.e., discontinuous. Indeed, altho techniques such as contour lines have difficulty representing them, these may be the most important features for many users. They certainly impact mobility.

4. There might conceivably be scale variance because of physical properties such as the finite strength of rock.
5. The data is heterogeneous; different regions have different statistics. For example, river basins occur mostly above sea level, while mid-ocean ridges occur under sea level. Some regions above sea level are karst terrain, with sink holes, while other regions have rivers.

6. Planetary bodies, such as the Moon, have different varied formation mechanisms, such as impact craters or large volcanoes.

7. There are long range correlations, such as river basins, that may extend from one side of a continent almost to the other ocean.

8. Terrain is often not spatially symmetric in the horizontal direction. Rivers’ headwaters, such as the Amazon’s, are often near one edge of the continent.

9. Neither is it symmetric in Z. There are many local maxima but few local minima, since they usually fill in and become lakes.

5.6 Find Fast heuristics.

A common Computer Science strategy is to study algorithms that are optimal in the worst case. Here, that is neither necessary nor sufficient. It is not necessary since optimizing sensor placement in the worst case may be take NP, i.e., exponential, time. On the other hand fast heuristics might require only polynomial time, with small exponents. This behavior is frequently seen, e.g., with bin packing and route planning. It is also not sufficient because the worst case may be much worse than the expected case.

5.7 Discover What We Certainly Know

(as close as we can get it.) The scene description is possibly approximate and partially erroneous. Nevertheless, some things may reasonably certainly be visible and other reasonably certainly be hidden. For instance, a sensor 10000 km hi can almost certainly see almost everything on almost one half of the earth’s surface regardless of errors and unknowns in the terrain data.

6 Proposed Timetable

We propose to start by, in parallel:

1. Researching feature-based morphological operators,

2. Detailing our drainage ideas,
3. Detailing our TIN ideas, and
4. Productizing our siting toolkit.

Then, in parallel

1. Researching the overdetermined PDE representation,
2. Productizing our TIN toolkit.
3. Detailing our feature-based morphological operators ideas, and
4. Productizing our drainage toolkit

And finally, in parallel,

1. Productizing our feature-based morphological operators toolkit, and
2. Tieing the pieces together into a unified terrain representation and operations toolkit.

7 Feedback?

We can’t do all of this, in the first year. Which areas are the most important to end users?
Geologically Correct Terrain Representations & Radar Siting
WR Franklin
Rensselaer Polytechnic Institute
wrf@ecse.rpi.edu, 703-447-7808
NGA Site Visit
RPI, June 27 2005

The Challenges

• Terrain data is voluminous.
• Existing compression methods destroy the most important features
• We need new representations to support operations such as multiobserver siting.
Where is This?

Answer
Examples of Other Errors

Goal

- Represent terrain (terrestrial elevation)
- Compactly; lossily
- Bias representations towards legal terrain
- Preserving usefulness for operations
  - Multiple observer siting with intervisibility
- Implement on the largest datasets.
- Study size / accuracy tradeoffs.
- Uncompression speed is more important than compression speed.
Impact on Defense

- More and higher-res terrain available.
- Processing power not keeping up.
- Inadequacies of current representations hinder military operations.

Demonstrated Need for Multiobserver Siting

- ARO STTR on TIN splines
- DARPA/SPO Robust Surface Navigation
Project Outline

• Representations
  – Feature-based operators like scooping
  – Overdetermined PDEs
  – Extensions to TINs

• Operations
  – Multiobserver siting with intervisibility
  – Drainage networks
First: Study Terrain Properties

- Many local max, few local min
- Long range order - rivers
- Elevation and slope discontinuities are common, and are very important for mobility and visibility

Which is land, which water?

You can answer this => there is unexploited structure.
Fourier Series

- Widely used
- Excellent for representing many physical phenomena, like vibrations.
- Quite unsuitable for terrain.
- They assume $C^\infty$ continuity
- The truncated series is too smooth
- ...and has many local minima

Project Outline

- Representations
  - Feature-based operators like scooping
  - Overdetermined PDEs
  - Extensions to TINs

- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks
Feature-Based Operators

- Use morphological operators that, to the extent reasonable, represent the physics of terrain generation.
- Initially: represent water erosion.
- Implement as a **scooping** operator.
- Scooping is just an example of possibilities.

Properties

- Feature-based operators are not linearly superimposable.
- Therefore computation will be compute-intensive.
- But reconstruction will be fast.
- Truncating a series produces a valid, tho less accurate terrain.
**Terrain Properties Modeled**

- Long-range, monotonically decreasing features (rivers).
- Many more local maxima than minima.
- Lack of symmetry between up and down.

**Procedure**

- The initial terrain is a block.
- A scoop is a 3D object, probably with circular vertical symmetry, such as a drill bit.
- A scoop follows a trajectory, from some start point to the edge of the terrain. This trajectory decreases monotonically in z.
- Unlike the summation of multiples of basis terms for a Fourier series, overlapping scoops do not sum.
- Successive scoop operations carve out the desired terrain.
 Simulation

 Details

• Incremental progress, based on experiments.
• First, refine scoops.
• Scoop along hand-drawn, then computed flowlines.
• Process the residual surface, and scoop it.
• Then, add other operators, like plop and scrape.
Parameters

- Many design choices are possible, including these:
- How many differently shaped scoops are allowed, such as spherical and conical. Scooping is best suited for eroded plateaus.
- Each shape’s dimensions, such as radius or angle.
- What type of curve the trajectory is, such as a cubic spline.
- The goal is to minimize the space needed to describe the terrain.

Scooping Status

- Still a preliminary idea.
- Greatest potential for longterm payoff.
Project Outline

- Representations
  - Feature-based operators like scooping
  - **Overdetermined PDEs**
  - Extensions to TINs

- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks

Overdetermined PDEs

- **Purpose:**
  - Interpolate from contour and point data to an elevation matrix
  - Do it better than previous methods.

- **Impact:**
  - Make large amounts of existing data more easily available at higher quality
Particular Advantages

- Can process both whole and incomplete contours.
- Can process isolated points.
- Infers local maxima inside topmost contours.
- Introduces no terracing artifacts on contours.
- Kidney shaped contours are no problem.

Method

- Set up a linear system whose unknowns are all the elevations, known and un-
- Two types of equations:
  - All points: $z_{ij}=(z_{i+1,j}+z_{i-1,j}+z_{i,j+1}+z_{i,j-1})/4$
  - Known points: $z_{ij}=h_{ij}$
  - (Each known point has 2 equations.)
  - Relative weights trade off accuracy and smoothness.
- Solve this overdetermined system.
Example 1

Accuracy vs Smoothness
Example 2 – Square Contours

Status

• John Childs masters project did a first version.
• Future:
  – Thorough evaluation of effect on siting of different accuracies on many datasets.
  – Extend testbed to larger datasets.
  – Productize.
Project Outline

• Representations
  – Feature-based operators like scooping
  – Overdetermined PDEs
  – Extensions to TINs

• Operations
  – Multiobserver siting with intervisibility
  – Drainage networks

TIN Current Status

• Existing program in C
• Input is square array of elevations, but program could be generalized.
• Processes 3603x3603 easily.
• For a given max error, it is 3 times faster than Arc/GIS, and
• The resulting triangulation uses 1/3 fewer triangles (for the same error).
Difficult Inputs are Easy

- Ridge lines are identified automatically.
- Discontinuities like road cuts are no problem.

Example

Original Terrain

Detail from TIN

Railroad embankment
Status and Possible Extensions

- It’s a standalone C++ program.
- Convert to subroutine for interfacing ease.
- Add higher-degree splines (though a preliminary test shows no improvement).
- Use nodes that are not points.
- Test new datasets.
- Interface to other RPI/Geo* components.

Project Outline

- Representations
  - Feature-based operators like scooping
  - Overdetermined PDEs
  - Extensions to TINs

- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks
The Operation Defines “Accuracy”

- Initially: RMS error
- Better: measure effect on operation, such as siting
- How much better is the solution computed using the best data than the solution using the more compact data?
- Formalizing this is tricky.

Project Outline

- Representations
  - Feature-based operators like scooping
  - Overdetermined PDEs
  - Extensions to TINs
- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks
Comparison Details

- Compute the operation, e.g., observer siting, on the original data.
- Measure $A = \text{area of joint viewshed}$.
- Site observers on the compressed data.
- Place those observers back on original data, and compute $B = \text{their joint viewshed}$.
- Quality of representation = $B/A$.

Details - 2

- A completely different observer siting is ok if it is just as good.
- Joint viewshed area, measured on the compressed data, can be significantly larger than when measured on the original data.
- ... proving that computing viewsheds should use the best data.
The Known Unknowns of Viewsheds

- Small changes in LOS interpolation cause large changes in visibility.
- One half of this cell has uncertain visibility.

Multiobserver Siting Steps

- Find approximate visibility index of every point in cell, using Monte Carlo sampling.
- Partition the cell into blocks and pick the best potential observers in each block.
- Using a greedy algorithm, select the best of the best observers.
- We have considerably studied the tradeoffs here.
**Enforcing Intervisibility**

- After the first best-of-the-best, add only new observers that are inside the joint viewshed of the previous best-of-the-best.

![Intervisibility Diagram](image)

**Effect of Intervisibility**

- This reduces the joint viewshed considerably.

![Intervisibility Effect Chart](image)
Reduced Resolution Effect on Siting

- Lowering horizontal resolution lowers observer siting quality.
- Lowering vertical resolution does not as much.
- Visibility, computed on lower resolution, is too high.

Experiments

Reducing horizontal resolution

Reducing vertical resolution

Visibility index vs. Resolution Reduction Factor
Status

- Standalone set of programs
- Needs optimizing, packaging, broader testing.

Project Outline

- Representations
  - Feature-based operators like scooping
  - Overdetermined PDEs
  - Extensions to TINs

- Operations
  - Multiobserver siting with intervisibility
  - Drainage networks
The Problem

- Computing hi-res drainage networks is slow.
- Partly caused by fake local minima that trap the water.

Solution

- We have developed a very fast connected component program.
- Use it to identify the separate fictitious basins and amalgamate them.
- Then solve a linear system for the water flow.
Connected Component Program

- Designed for 3D cubes of bits, over 500x500x500.
- Does 2D as a special case.
- Processing a binarized 18573x19110 image took 25 CPU seconds on a 1600 MHz laptop.

Impact

- Better prediction of flooding and river widths, which affects mobility.
- Tests quality of our previous terrain representations.
Status

• Connected component program works.
• Must modify it for drainage.
• With local minima removed, solving for flow looks easy in Matlab.
• Tests on large datasets will probably find unexpected things.

Project Outline

• Representations
  – Feature-based operators like scooping
  – Overdetermined PDEs
  – Extensions to TINs

• Operations
  – Multiobserver sitting with intervisibility
  – Drainage networks
Interactions: Nodes

- One of our strengths is that the various components may interact.
- Node producers: TIN, siting
- Node consumers: PDE, TIN
- Combine them.

Interactions: Ridge Inference

- PDE infers hilltops.
- If the posts are too low-res to contain hilltops and ridges, use PDE to infer them, then feed result to TIN.
- Challenge: The various programs’ I/O must be made consistent.
Themes

1. Nonlinearity is powerful.
2. Large memory is now available.
4. ... But not always.
5. Respect the terrain.
6. Find fast heuristics.
7. Discover what we certainly know.

Our Players

• Dr. Randolph Franklin
• Dr. Caroline Westort,
• Metin Inanc, PhD student
• Some undergrads during summer
• ESRI?
• Another PhD student?
Major Challenge

- It's a big jump from a few separate demo programs to a unified reliable system.
- F Brooks in *Mythical Man Month* says that this is a cost factor of 9.

Major Required Input

- Which parts are more interesting?
Here is what we've been doing on the Geo* award at RPI since the contract was awarded.

1. RPI is moving to offer Dr Caroline Westort a Research Assistant Professor position. She will be doing various things, such as radically new terrain representations, such as scooping.

2. Metin Inanc, a PhD student, is studying aspects of the TIN representation, including using higher order splines. An initial implementation of this did not improve the accuracy. Perhaps terrain is not generally continuous at a higher order. That would align with some arguments I've been making.

3. Jared Scheuer, a junior, has been studying aspects of my overdetermined Laplacian representation. He's been measuring the accuracy of either subsampling the surface, or of having a region of missing data, and using this technique to fill in the missing data.

4. Rositsa Tancheva, another undergrad, has been researching non photorealistic rendering (NPR) in computer graphics, in order to apply that to terrain. This is exploratory work that may lead to another new representation.
5. John Shermerhorn, another undergrad, is also looking at some terrain issues, such as creating a testbed in which we can try our ideas.

After the summer, most of the undergrads will choose to be off the project to do classwork, but Dr Westort will be on board.

6. There is also a new grad student, Zhongyi Xie, coming to RPI whom I am sponsoring as a TA, and whom I may pick up as an RA, depending on how he works out.
Progress Report - RPI Geostar Project - Aug 2005

WR Franklin
ECSE Dept, 6003 JEC, Rensselaer Polytechnic Institute, 110 8th St, Troy NY 12180
(518) 276-6077, wrf@ecse.rpi.edu

1. Following on a question posed at the site visit last June, we studied interpolating large holes in the terrain data. See this report. The code is in Matlab, and is quite brief. (The creativity lies in the idea and in the careful design.) If there is interest, we can perform further tests on NGA-supplied data, and/or document the code for your use.

2. Metin Inanc is working on our overdetermined Laplacian PDE algorithm, which can be interpolate missing data, and smooth existing data, in the following ways:
   a. Combine the overdetermined concept with the thin plate PDE.
   b. Implement a more efficient solution technique, to replace the Matlab built-in least-squares solver.
   c. Investigate combining overdetsol with the TIN as follows:
      i. Use our TIN program to select important points on the terrain.
      ii. Use overdetsol to interpolate a surface from them.
      iii. Evaluate that surface's quality, initially by its RMS error compared to the original surface.

The information content in this representation is the set of important points. (Indeed, since the order of the points is
unimportant, the info content is a little smaller than the sum of the bits in their coordinates.) If a good surface can be reconstructed from a set of reasonable size, then we have another compact terrain representation algorithm.

3. Dr Caroline Westort is now officially on the RPI faculty as a Research Assistant Professor supported by this contract. Her initial tasks will be as follows.
   a. to combine her digital terrain modeling ideas, dating back to her thesis.
   b. to search out good ideas and possible collaborators in the community.

4. Dr Frank Luk has agreed to provide his expertise in numerical techniques, initially to optimize our overdetermined PDE algorithm. Prof Luk, in the Computer Science department here at RPI, is a former chair of that department, and is an internationally renowned expert in Scientific Computation.

5. The three summer students have left. When possible, we will incorporate their work into this project.

6. We may take on an undergrad to study water flow this semester. He would apply my fast connected component program to identifying watersheds in order to remove the interior ones. This step is necessary in computing hydrology. Current techniques appear to be slow, partly because they often require external storage. We think we can do better.

7. I (WRF) am thinking about scooping, and other new techniques.
Progress Report - RPI Geostar Project - Sept 2005

WR Franklin
ECSE Dept, 6003 JEC, Rensselaer Polytechnic Institute, 110 8th St, Troy NY 12180
(518) 276-6077, wrf@ecse.rpi.edu

We continue to push in parallel several themes in terrain representation. Here is a more detailed description of our motivation, of their innovative aspects, and of expected future challenges.

Terrain Reconstruction by Scooping

This subproject has the largest potential payoff, for several reasons. It will be able to represent discontinuities (cliffs), and its nonlinearity is powerful. Dr Westort is creating an initial testbed in Matlab to help us understand the idea. This initial testbed will be of the nature of a terrain reconstruction assistant. The user will specify the terrain, the X-Y paths that the scoop (also called a blade) will follow, and the blade's circular cross-section. The system will run the blade along the path at the lowest elevation that does not cut into the terrain. It will show the error at each post and report the average and max error. We anticipate demonstrating a prototype at the review meeting in San Diego.

This testbed will provide a facility to gain experience with different blade shapes, including variable shapes. That experience will allow us to freeze some decisions, such as whether to use variable shapes, or not. The former choice will require fewer cutting paths, but with the latter choice, each path will take less space to represent. An analogy to splines would be, what is the appropriate polynomial degree to use?

Cutting paths are appropriate for canyon-like terrain. However,
reconstructing, say, a mesa, might be better done with other operators. Designing such operators will be the next step, after we can reconstruct eroded terrain.

Postprocessing the reconstructed terrain with a smoothing operator might improve the accuracy at the cost of adding only one bit to the size of the representation ("apply the builtin smoother"). Smoothing will effective if the terrain satisfies some smoothness criterion, which contradicts our design principle that terrain is often discontinuous. Nevertheless, for completeness, this should be checked.

Finally, after obtaining a sufficient understanding of this idea, we might think about formulating and proving some theorems about it.

**Interpolation With Overdetermined PDEs**

The central idea is to reconstruct the terrain by approximation from a set of known points. (Approximation is a more accurate term than interpolation since the surface may not pass through the given points.) There are two steps: determining what points to use, and then fitting the surface. With a given fixed approximation algorithm, the storage required to represent the surface is the space needed to store the points, suitably compressed.

The novelty of this subproject has two components:

1. **Combining two previously unrelated techniques.** Selecting more important or outstanding points, instead of using, say, a regular subgrid of points, seems reasonable (though we plan to test this). Our initial technique for selecting points on the surface uses a Triangulated Irregular Network (TIN). We can TIN a 10000x10000 tiled DTED level II surface. Although the TIN is a classical technique, it does not seem to have been applied in this way, perhaps partly because no one else can easily TIN that large a dataset.

2. **Using an overdetermined PDE to reconstruct the surface.** The novelty resides in our extension, described in the proposal and other places, of the classical application of PDEs for interpolation. This produces smoother surfaces with no visible
We expect this subproject to produce results more quickly than the scooping subproject. It also has slightly different applications, such as filling holes of missing data. Last month's report described filling holes of radius 100. We are now working on filling the large irregular hole in some DTED level II data supplied at the March kickoff meeting in Savannah.

Metin Inanc is working on this subproject. Prof Luk is researching its numerical analysis aspects, primarily how to solve these large overdetermined sparse systems. Currently, we can process up to about a 400x400 grid, depending on which PDE is used. Currently we use either a Laplacian heat flow, or a thin plate PDE. The former is faster, while the latter is smoother.

Next, Prof Luk will study the accuracy of slopes in the reconstructed surface. One could imagine a reconstructed surface with high frequency noise, so that the elevation was accurate but the delta between adjacent posts was quite wrong. Lossy compression techniques designed for image processing seem to do that sometimes. However, our overdetermined PDE technique reconstructs a smooth surface. If anything, it might underestimate the true slope. We will ignore that until we know what "true slope" means when the surface is sampled only every 30m or so.

Here is another possible way to select the points to approximate.

1. Pick S, an initial set of points, perhaps a coarse grid with extra points near the boundary.
2. Approximate a surface through them.
3. If the max error is less than a desired tolerance, then exit.
4. Otherwise, find the most erroneous points on the reconstructed surface.
5. Insert them into S and go back to step 2.

The most obvious concern is that this is much too slow. Computing one surface may take hours on a small dataset, depending on factors...
like which PDE is being used. Depending on how many points are inserted in each iteration, hundreds of intermediate surfaces might need to be computed. That is why another method of generating the points (i.e., the TIN) is desirable.

**Upcoming Decisions**

Here are several things we intend to do soon.

1. Metin is transitioning to a 64-bit machine because of insufficient address space when working with datasets over 100M points. That is nontrivial because some application software doesn't yet work well in this regime. That includes C++ iostreams and possibly Matlab.

2. Zhongyi Xie, a new grad student will initially be assigned to helping Metin, and then probably to the hydrography application.

3. Based on feedback from the June site visit to RPI, we have delayed awarding the subcontract to ESRI until this project was more advanced. However, we will decide this soon. ESRI needs to be given something relatively mature to productize. Our multi-observer siting work is the obvious choice. The novelty of this work is that it goes beyond finding individual viewsheds and visibility indexes to optimizing a set of observers as a set.
Our overdetermined PDE terrain approximation scheme requires a source of posts from which to interpolate. Posts that are "outstanding" in some sense appear to be a reasonable choice. Such points might be those selected by our Triangulated Irregular Network program, TIN. Therefore we are doing the minimal amount work on TIN necessary to suit it for this purpose.

1. We have adapted TIN to process datasets of size 10800\times10800, formed from combining 9 DTED level II datasets from the DVD obtained at the Savannah kickoff meeting in March 2005.

2. TIN runs on a 32 bit Xeon machine running Linux configured to allow 3GB of user address space per process. Although TIN also runs on Windows, the largest feasible dataset is smaller, apparently because Windows, even with the \(/3G B\) switch, doesn't allow large contiguous blocks of storage.

3. We think that the elevation range of the dataset is 0 to 3625, when negative values are replaced by zero, but might possibly be misinterpreting the data.

4. If the original dataset is stored with 2 bytes per point, the \(10800^2 = 116,440,000\) points take 233MB (following the standard that \(1M = 1000000\)). Only 2 bytes are necessary since the points are in a regular array, and so only the Z values need be explicit.

5. For the output size, we assume 6 bytes per point. Since the points are irregular, X, Y, and Z need to be stored.

6. The topology (i.e., the triangles) need not be stored since it is implicit since the points form a Delauney triangulation.

7. In practice, both the input and the output files would be compressed. Assuming the compression quality would be similar in both cases, we can ignore it when computing the ratio of storage improvement.

8. Our "elevation accuracy" is usually the maximum elevation error. Occasionally, there may be a few points with greater errors. The reason is that occasionally splitting a triangle into 3 subtriangles increases the error. If desired, we could modify TIN to spend the time to be certain about the max error.
10. Here are two sample runs on the data:

<table>
<thead>
<tr>
<th>Elevation accuracy</th>
<th>Number of output triangles</th>
<th>Number of output points</th>
<th>Output/input points, %</th>
<th>Execution time, CPU secs</th>
<th>Output size</th>
<th>Output/input size, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>157,735</td>
<td>79,382</td>
<td>0.07%</td>
<td>1292</td>
<td>476KB</td>
<td>0.2%</td>
</tr>
<tr>
<td>10</td>
<td>5,320,089</td>
<td>2,661,466</td>
<td>2.3%</td>
<td>1638</td>
<td>16MB</td>
<td>7%</td>
</tr>
</tbody>
</table>

11. When these points are fed into our overdetermined PDE approximation, we expect the errors will improve.

Copyright © 1994-2006, W Randolph Franklin (WRF) You may use my material for non-profit research and education, provided that you credit me, and link back to my homepage.

Geologically Correct and Compact Terrain Representation
DARPA-NGA/Geo* Review
San Diego, November 4 2005

W Randolph Franklin, Caroline Westort, Frank Luk, Metin Inanc, Zhongyi Xie
Rensselaer Polytechnic Institute
mail@wrfranklin.org, 703-447-7808
http://wrfranklin.org

Team

- W Randolph Franklin, RPI, Center for Subsurface Sensing and Imaging Systems – leader, research
- Caroline Westort, Research Assistant Prof, RPI - terrain sculpting.
- Frank Luk, Prof, CS Dept, RPI – numerical analysis
- Metin Inanc, PhD student, RPI
- Zhongyi Xie, new PhD student, RPI
- COL Clark K Ray, about to leave USMA, occasional advising
- Duane Niemeyer, ESRI – develop ArcGIS applications and transfer technology into NGA programs
Team changes since kickoff

- Prof Westort and Prof Luk joined this fall.
- COL Ray is mostly unavailable (and may be in Afghanistan for a few months).

Prof Caroline Westort

- Joined project as a Research Assistant Prof.
- Education: Harvard, Zurich
- Thesis: *Methods for Sculpting Digital Topographic Surfaces*
- Organized *Digital Earth Moving* conference
- Worked at Modeling and Simulation Div of TEC
- Most recently: U Va in Landscape Arch
- Organized GPS-Guided Earthmoving workshop at ASLA in Oct.
Prof Frank Luk

- Joined team for his expertise in numerical computing.
- Education: Caltech, Stanford
- Former Chair, RPI/CS
- Expertise: parallel and distributed scientific computing

Goals of this project

- Represent terrain (terrestrial elevation)
- Compactly; lossily
- Bias representations towards legal terrain
- Preserve usefulness for important applications
- Study and perhaps combine several methods
- Implement on the largest datasets.
- Study size / accuracy tradeoffs.
- Uncompression speed is more important than compression speed.
**Project Outline**

- **Terrain Representations**
  - Sculpting
  - Overdetermined PDEs
    - SPIHT wavelets
    - Singular Value Decomposition
    - Image Infilling
    - Higher order splines

- **Terrain Operations**
  - Filling in missing data
  - Multiobserver siting with intervisibility
  - Hydrology
Metric

- RMS elevation error
- Max absolute elevation error
- "X" on slope
- Effect of errors on terrain operations like multi-observer siting

Big idea - Terrain sculpting

- Goal: Nonlinear operators that approximate geological processes
- Initial operator: scoop (aka blade)
- It removes material in a milling operation.
- Plan for future additional operators.
Sculpting metaphor

- Michaelangelo looks at a block of marble
- He sees David inside the stone.
- He removes all the stone that is not David.
- Hypothesis: the list of his sculpting actions is an efficient description of David.

Terrain sculpting details

- Caroline Westort is doing this
- Create a Matlab testbed as a semi-intelligent assistant
- Test sculpting some real terrain
- User specifies 2D path for scoop.
- Testbed scoops along path, showing resulting terrain, difference wrt real terrain, and stats.
**Sculpting properties**

- If scoop's elevation decreases monotonically until scoop reaches the edge of the world, then interior minima are impossible.
- Cliffs are natural.
- Hydrology is modeled well.

---

**Version 0**

- User specifies 3D path
- System cuts it into a block.
**Version 1**

- User selects test terrain and scoop shape.
- User draws a 2D path on terrain, perhaps following a river.
- System drags scoop along path, just touching terrain.
- System shows error.
- Repeat.

**Version 2**

- System picks path automatically, perhaps along biggest river.
- After each step, recompute hydrology and repeat.
Version 3

- Experiment with varying scoop size and shape.
- Tradeoff: This uses fewer paths but each path takes more bits to describe.

Version 4

- Consider terrain addition and moving operators, not just terrain removal.
- Now, the scoop becomes a bulldozer blade.
GPS guided bulldozer

- Separate file because of size

Sculpting automation

- Search the space of terrain operators for optimum combo (or, as good as we have time to search for).
- This is very slow, but the reconstruction is much faster.
- Finally compress the description, initially with bzip2.
New science

- Previous CS-like representations have used ignored the real terrain challenges.
- Previous geography terraforming research has ignored the CS challenges.
- We’re fusing CS and geography competence.

Deliverable

- Better terrain compression prototype, preserving drainage and discontinuity properties.
Status

- Ramping up; working on version 1.
Overdetermined PDE

- Competing representation technique
- Goal: Recreate terrain by approximate interpolation from a set of data points.
- Our approximation technique is an *overdetermined PDE*.
- This combines good accuracy with good smoothness, and allows a point-by-point tradeoff.

Overdetermined PDE properties

- This does not prevent local minima, nor allow cliffs.
- However, it is new and interesting and likely to produce results sooner.
- It handles conflicting data.
- It can smooth the data.
- Accuracy and smoothness can tradeoff
- ... locally
**Method**

- Set up a linear system whose unknowns are all the elevations, known and un-.
- Two types of equations:
  - All points: \( z_{ij} = (z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1})/4 \) (for Laplacian).
  - Known points: \( z_{ij} = h_{ij} \)
  - Each known point has 2 equations.
  - Relative weights trade off accuracy and smoothness.
- Solve this overdetermined system.

**New science**

- Using an overdetermined system.
- This lets information flow across known data, and allows accuracy-smoothness tradeoffs.
- We’re developing the NA techniques to solve these large sparse systems.
- Dr Luk is researching least squares ADI method to minimize fill-in.
- Also examining boundary value issues.
Choice of PDE

- Initially: Laplacian or thin-plate PDE.
- Laplacian is faster; thin-plate smoother.

Status

- Solving sparse overdetermined linear systems is slow and requires much memory than solving a square matrix.
- We can do a 400x400 grid in Matlab on a PC.
- Our goal is 10000x10000.
- Fitting together pieces of smaller solutions looks wrong.
Deliverable

- A working prototype

Selection of data points

- The input points for the terrain approximation should be “important” in some sense.
- We’re using points output from a TIN approximation.
- We can TIN a 10801x10801 matrix of elevation posts on a PC w/o external storage in about ½ hour.
Selecting “Feature” points

- Points on peaks and ridgelines may be more important.
- No need for manual identification.
- Our TIN program selects them automatically.

TINning $10^8$ posts

- 9 DTED II cells combined => 10800x10800
- Z range: 3600
- External data structure: none
- Key idea: very efficient internal TIN representation.

<table>
<thead>
<tr>
<th>Elevation accuracy</th>
<th>N output triangles</th>
<th>N out pts / N in pts</th>
<th>Exec time, CPU secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>157,735</td>
<td>0.0007</td>
<td>1292</td>
</tr>
<tr>
<td>10</td>
<td>5,320,089</td>
<td>0.023</td>
<td>1638</td>
</tr>
</tbody>
</table>
Combining techniques

- The optimum terrain representation may be a combo of various techniques.
- Perhaps
  - Start with a PDE
  - Then edit result with terrain sculpting operators.

Possible multistep refinement

- Given an overdetermined PDE surface produced from a set of points
- Find the worst points on that surface
- Insert them into the point set
- Find a better surface
- Repeat
- Limitation: finding one surface takes hours.
**Other possible methods - 1**

- Higher order TIN
  - Apparently no better than piecewise linear
  - i.e., terrain is not that continuous.
- SPIHT wavelets
  - Done that 10 years ago.
  - Reasonably good.

**Other Possible Methods - 2**

- Singular value decomposition
  - Preliminary test shows it’s reasonable.
  - Presumably limited by not using any special terrain info.
  - (Unless that special info doesn’t help!)
  - More tests necessary.
Other Possible Methods - 3

- Image infilling
  - New hot topic at SIGGRAPH
  - Would a technique designed for images help for terrain?
  - Tests necessary.

Terrain applications

- Filling in missing data
- Multi-observer siting
- Hydrology
Interpolating missing data

- An application of overdetermined PDE.
- Assume a matrix of known elevations, except for an unknown circle of radius 100.
- We fill in the missing data better than all the Matlab built-in methods (not that that’s hard to do).

Inferring interior maxima
Comparing methods

Interpolating nonuniform posts

- Constructs smooth surface from nonuniform data
- Infers extrema
- Data shadows other data
  - Not characteristic of most methods
  - However, very desirable
- No fixed scale
  - Also not characteristic of other methods, but good
**Data deconfliction & smoothing**

- Overdetermined PDE handles inconsistent data.
- Can tradeoff accuracy and smoothness, point by point if desired.
- Non conflicted data can be smoothed.

---

**Multi observer siting**

- **Description:** Find a set of observers jointly to cover most of a cell.
- **Status:** Working research-quality code.
- **Next:** Productize.
- Also various open research topics.
CONTENT WARNING

Following is our token slide copied from our Savannah kickoff presentation

With or w/o intervisibility
Contrast to existing tech

- We represent viewsheds as large bitmaps.
- No size limit.
- R=200 easy.
- Boolean ops on viewsheds, testing point inclusion much easier than for polygonal viewsheds.

Interpolating LOS between posts

- Possible research
- Motivation: in one test, we tried various interpolation methods (min, max, linear)
- ½ of all the targets changed visibility
Hydrology

- This is both an application and a source of sculpting paths.
- Intuitive idea:
  - Rain falls on terrain.
  - All the water entering an interior pixel flows out to the lowest neighbor.
- Problem: lowest neighbor may be higher than here, so no water flows out.

Problem of interior minima

- Some are caused by data measurement errors,
- Others by the finite resolution of the terrain sampling.
- They are so frequent that they will trap and inhibit all long rivers.
- They must be removed.
- Current methods are very slow.
**Complexity of interior minima**

- If they were simple isolated posts, a median filter would suffice.
- But, they may be large regions.
- Gradually raising the water level until the dam overflows is quite slow.
- Similarly, a large flat region is problematic (and raising the water level doesn’t work here).

**Our solution**

- Use our fast connected component program to identify drainage basins.
- Edit post elevations of interior basins.
- Recompute water flow, by solving a large sparse linear system.
- We hope it won’t require external data storage => much faster.
**Fast 3D connected components**

- Designed to find connected components in a 3D binary universe.
- Test on one 512x544x544 universe:
  - 151,519,232 voxels, 76,073,721 empty.
  - 15 seconds CPU on a 1600 MHz Pentium to find all connected components, and volumes and surface areas.
- There are 534,723 6-connected, or 223,339 26-connected components.

**Fast 2D connected components**

- Trivial special case for the 3D program.
- Test: 18573 by 19110 pixels.
- Time: 25 secs on a 1600Mhz Pentium.
Status

- Connected component program exists.
- No implementation work yet on applying it to hydrology.

Terrain test data

- We start with level I USGS data such as Lake Champlain West
- Then try DTED level II data from DVD obtained at Savannah kickoff
  *(but that’s not big enough)*
- Then tile 9x9 DTED level II cells, giving 10800x10800 posts.
Quality metric

- Initially, minimize maximum elevation error.
- Also, measure slope errors in reconstructed terrain.
- Note that both our methods should reconstruct more accurate slopes than Fourier series.

Our greatest need

- Meeting end users
Status reports and deliveries

- http://wrfranklin.org/Research/geostar-reports/

Next 6 months: parallel streams

- Pursue several parallel streams, with subtasks
- Scooping (long term)
  * Extend testbed
  * Think about more general operators
- PDEs (midterm)
  * NA techniques for larger solutions
  * Test point generation techniques
- Hydrology (short term)
  * Adapt my fast connected component program
- Multi-observer siting
  * Productize with ESRI
Progress Report - RPI Geostar Project - Nov 2005

WR Franklin
ECSE Dept, 6003 JEC, Rensselaer Polytechnic Institute, 110 8th St, Troy NY 12180
(518) 276-6077, wrf@ecse.rpi.edu

Here is what has happened on RPI's part of the DARPA/DSO/Geo* project during November.

Contents:

1. Followup on San Diego Meeting
2. Scooping, Ctd
3. SVD
4. Overdetermined PDE
5. Combining Techniques
6. Publicity About This Project
7. What We'd Like From You
8. External Visitor
9. What's Next
1. Followup on San Diego Meeting

Here are answers to some questions that were raised during our presentation.

1. Q: Our missing data interpolation scheme can infer a local maximum. How is that possible since the Laplacian PDE cannot do that?

   A: We're using an *overdetermined* version of a Laplacian PDE. The unknown points are only approximately the averages of their neighbors. Our technique has extra power that the exact Laplacian does not have.

   Another advantage of the overdetermined method is that, if the gaps between contour lines are interpolated, then the contour lines are not visible in the resulting surface. In contrast, when the exact Laplacian is used, the resulting surface looks like a loose cloth draped between the contours. Indeed, we devised the overdetermined technique precisely to avoid these limitations, which are common to most existing interpolation methods.

2. Q: Why don't we just use image infilling?

   A: Image infilling is an image processing technique primarily designed to remove an existing object from an image, while filling in the gap with a continuation of the background so that it isn't obvious that the missing object was ever there. One application is to remove someone from a photograph. This has been a hot topic at, e.g., SIGGRAPH, for a few years.

   Instead of removing an existing object, we wish to infer an existing object. Also, the criteria for producing an image that looks good are not identical to the criteria for reasonable terrain. Our technique has other advantages. For instance, it can utilize isolated elevation points, when they are available.

   Finally, in order to move forward, at some point we have to stop...
looking sideways, and move forward.

2. Scooping, Ctd

As a refresher, we will represent terrain by a sequence of drill operations applied to a block. Each drill operation removes some material, until only the desired terrain remains. Each drill operation is specified by

1. Which drill to use, and
2. Where to apply it, (x,y,z).

The final material removed is the union of the volume removed by each operation. This is a major difference between our approach and any competing technique that sums linear multiples of a set of basis functions. Their combining operator is addition; ours is union. We are working on the following fronts.

1. What are the best drills to use is a major question. Flat? spherical? something else? I have produced a Matlab testbed to do the following:
   a. Take some terrain, and
   b. definitions of a number of drills. A drill is defined by the elevation of each point on its face about a reference plane. We assume that the drill is being applied as with a 3 axis milling machine, so that this completely defines a drill. We have tested up to 16 drills at once.
   c. In a greedy optimization manner, select the drill and location that would remove the greatest volume.
   d. Repeat.

2. Zhongyi Xie will start experimenting with different drills.

3. It might be efficient for a drill to have more parameters than location. E.g., its surface might be rotatable to different
orientations and angles. (The alternative would be to have many simpler drills, each with a specific orientation and angle.) Metin Inanc is working on this.

4. The drill operations might not be independent, but probably would follow the drainage network. (We are not doctrinaire, and will do this only if it is more efficient than not doing it.) Caroline Westort is studying this aspect, initially by researching drainage.

5. As part of this, we have written the first version of a hydrology flow computation program in Matlab. Assuming that all the water entering every post flows out to its lowest neighbor is easy. Handling the local minima, which trap the water flowing into them and so inhibit long rivers, is a harder. As described in the proposal, we are planning to use our fast connected component program here.

3. SVD

Singular Value Decomposition is an alternative representation method the Frank Luk thinks might prove competitive with scooping. Dan Tracy, his grad student, is experimenting with this.

4. Overdetermined PDE

This is another representation with quite different properties than scooping. It compares to scooping as follows:

1. It doesn't handle perfect cliffs, but does do steep slopes w/o any Gibbs ringing.

2. It fits a surface near data points that can be isolated points or strings or closed loops of points.

3. It can process overlapping and inconsistent data, fitting a surface near the various points according to their relative weights.

4. It is well suited to interpolating regions of missing data, partly
because its resulting surface has continuous slope across the boundary between of the missing region, so that that boundary is invisible.

5. It doesn't yet scale up so well to large datasets as does drilling. Approximately 700x700 cells work, slowly. The problem is that beyond this point the default Matlab solution technique for overdetermined linear systems requires too much memory for a 32-bit processor. We are acquiring a 64-bit quad Opteron with 64GB for larger tests.

That third property above may be of use to you as follows.

1. Load a UAV with a lo-res terrain database for a large region.
2. During flight, update selected subregions with better data.
3. Our overdetermined PDE will seamlessly merge the small hi-res data into the large lo-res data.

If this is of interest, tell us more about your overlapping data characteristics, and give us some data.

At San Diego, interest was expressed in property 4 above, and we were instructed to perform more rigorous tests of the following form:

1. Take a complete array of terrain elevations.
2. Delete a region.
3. Use our overdetermined PDE to fill in the missing region.
4. Compare our computed terrain with the real terrain that was deleted.

We have done so, with the favorable results to be written up soon.

5. Combining Techniques

Our ultimate recommendation may well be a combination of our various techniques, perhaps as follows:
1. Reconstruct an approximate surface with scooping.
2. Smooth it with an overdetermined PDE.
3. Fill in any local minima with our fast connected component program.

Note that the last two steps do not increase the information content of our representation. That is, if they are a fixed part of our reconstruction engine, having them operate adds zero bits to the representation of the terrain.

6. Publicity About This Project

After clearing it with DARPA, RPI issued a press release about DARPA's award to RPI, and several blogs picked it up. Some of these links may eventually become invalid. In addition I was interviewed briefly on WGY 810 radio on Nov 3 or 4.

1. RPI
2. zdnet
3. surfwax
4. defensetalk
5. acm

7. What We'd Like From You

1. Terrain data with cliffs. That's our strong point and what we said we'd do.
2. For any terrain data, we'd like to get it at a little higher precision than it is accurate. E.g., if
3. Feedback on whether to pursue using the overdetermined PDE to overlay and combine different terrain data. This is a
serendipitous side-effect of the technique, and so would be easy to demonstrate, if you're interested.

8. External Visitor

Prof. Marcus Alvim Andrade from the Computer Science Department of Federal University of Viçosa (Brazil) spend 10 days visiting us to discuss hydrology and terrain representation algorithms.

9. What's Next

I need to clean up our multi-observer siting code to get it into a useful state for future work. This was joint work by Christian Vogt, a masters student who graduated in Dec 2004, and myself. Then we can use a more sophisticated error metric. That would be as follows.

1. Site a set of observers on the original terrain so as jointly to cover as much as possible. Let their joint visibility index be \( V_1 \).

2. Site a set of observers on our lossily compressed terrain, after reconstruction, so as jointly to cover as much as possible.

3. Transfer those observers' locations back to the original terrain, and compute their joint viewshed. Let their joint visibility index be \( V_2 \).

4. Compute \( V_2/V_1 \). The closer that this is to 1, the better.

Why do we transfer the observers computed on the reconstructed terrain back to the original? This is because we are interested in surveilling the original terrain. Also, we have observed that when viewsheds are computed on approximate terrain, the results can contain a systematic bias. This was discussed in our 2004 ISPRS poster and some unpublished papers mentioned here.

We are on track to compress the terrain on the DVD received at Savannah in March 2005 to 10% of the space it occupies on the DVD, while maintaining a reasonable error.
Geologically Correct and Compact Terrain Representation
Status, Jan 2006

W Randolph Franklin, Caroline Westort, Frank Luk, Metin Inanc, Zhongyi Xie
Rensselaer Polytechnic Institute
mail@wrfranklin.org, 703-447-7808
http://wrfranklin.org

Mostly New Material

- This slide set builds on our Nov 2005 San Diego presentation, and has little overlap.
- Our periodic progress reports have more material:
  http://nga:24nov03@www.ecse.rpi.edu/Hompages/wrf/Research/geostar-reports/
- Other results remain to be written up.
Team Summary

- Prof Randolph Franklin – helping everyone
- Prof Caroline Westort – strategizing about scooping
- Prof Frank Luk – numerical analysis advice
- Metin Inanc – scooping etc
- Zhongyi Xie – ODETLAP
- Dan Tracy – SVD approximations

Test Data Used

- USGS 1° DEM: Lake Champlain W.
- DTED level 2 data from March 2005 kickoff meeting.
- SRTM DTED level 2 data from Jan 2006 CDs.
ODETLAP Status

- This is our overdetermined Laplacian PDE.
- Processing 400x400 arrays of elevation posts.
- Initially sample that with a subarray of regularly spaced points, every K points in each direction.
- When computing a complete surface from the sample points, parameter R trades off accuracy vs smoothness.
- Observe tradeoff of data size versus K, R on a mountainous region of the USGS Lake Champlain W level 1 DEM.
- Next step: use outstanding points from TIN, which should be better than regular points.

TIN Status

- We can process 10800x10800 arrays of posts in ½ hr on PC w/o external storage
- Dataset formed by catenating 9 3601x3601 cells from data from the Savannah March kickoff meeting.
- Elevation range: 3600.

<table>
<thead>
<tr>
<th>Elevation accuracy</th>
<th>N output triangles</th>
<th>N out pts / N in pts</th>
<th>Exec time, CPU secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>157,735</td>
<td>0.0007</td>
<td>1292</td>
</tr>
<tr>
<td>10</td>
<td>5,320,089</td>
<td>0.023</td>
<td>1638</td>
</tr>
</tbody>
</table>
**TIN ctd**

- "Feature" points on peaks and ridgelines may be more important.
- Our TIN program selects them automatically; no need for manual identification.
- The points selected for the triangulation are assumed to be important, and will be fed into other methods, like ODETLAP.
- Note that the TIN is a piecewise linear triangular spline. Preliminary experiments with a higher degree spline showed no improvement, and so were discontinued.

---

**Scooping Status**

Several subprojects:

- 3-axis milling machine experiments with set of simple drills.
- Complete cover test with parameterized sloped drills.
- Theoretical thinking about how scooping is different from, e.g., wavelets.
Drilling Test on Lake Champlain W Cell

- 1201^2 posts.
- Initial volume of extra material: 1.9E9
- Remaining after 10K steps: 4.2E7.
- Frequency of use of each drill type
  → This is very preliminary.

<table>
<thead>
<tr>
<th>Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square, D=1</td>
<td>0</td>
</tr>
<tr>
<td>Square, D=3</td>
<td>35</td>
</tr>
<tr>
<td>Circle, D=5</td>
<td>2</td>
</tr>
<tr>
<td>Circle, D=10</td>
<td>129</td>
</tr>
<tr>
<td>Circle, D=20</td>
<td>9834</td>
</tr>
<tr>
<td>Parabola, size 5</td>
<td>0</td>
</tr>
<tr>
<td>Parabola, size 10</td>
<td>0</td>
</tr>
<tr>
<td>Parabola, size 20</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 5</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 10</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 20</td>
<td>0</td>
</tr>
</tbody>
</table>

Sloped Drill Details

- Purpose: to better understand scooping, while initiating experiments in slope-preservation while lossily compressing.
- Underlying assumption: little long range correlation of elevation or slope.
- Method:
  * Cover the cell with a regular tiling of drills of size 5.
  * Each drill face’s slope is adjusted with SVD to best fit the data.
- Test data:
  * USGS Lake Champlain W cell.
  * SRTM w124 n38 dt12
- Next: Greedily fit from a set of possible drills of various sizes, to reduce max error.
- Later: follow along drainage paths.
Sloped Drill on Lake Champlain

- Test data: USGS Lake Champlain W cell. (Data range: 1576).
- Over 50% of elevation posts are exactly represented.
- Max error: 101, but that was rare.
- Input data size: 1442401 numbers.
- Output data size: 172800 numbers.
Sloped Drill on SRTM Data

- W124N38 dataset.
- Elevation range: -12 to 1022.
- 3601x3601 elevation postings, ¾ ocean.
- 99.97% of the postings are within 16m of the original.
Hydrology Status

- Can compute workflow ignoring the problem of local minima (but that’s easy).
- However, local watersheds trap all the long rivers.
- Fast connected component program works.
- To do: use the latter to identify local watersheds so they can be removed.

Multiobserver Siting Status

- This SW works, though it’s complicated and messy.
- We hope for RPI and ESRI to agree on the subaward this week.
- Then ESRI will productize it in ARC-GIS.
With or w/o intervisibility

Filling in Missing Data

- ODETLAP can be used to fill in missing holes, although this isn’t its main purpose.
- We extensively tested it on the Lake Champlain W cell:
  - 20 different locations
  - Circles of missing data of radius 5, 10, 20, 40
  - Emphasize either accuracy or smoothness.
  - Plot contours of the real data and of the interpolated data
  - Histogram the error
- Next slide shows one experiment.
- Note that, even though the (fine) contours appear different, the average error may be small.
Publicity

- RPI announced our award in a press release.
- 4 blogs picked it up:
  - ACM Technews.
  - Defencetalk
  - Surf wax
  - Zdnet
- Current Rensselaer alumni magazine has $\frac{1}{2}$ page on this project.
**Future Strategy**

Our ultimate solution is expected to be a sequence of all of our techniques:
- Use TIN to find important points.
- Use them to guide scooping.
- Smooth the resulting surface with ODETLAP.
- Remove any local minima with our hydrology programs.
Geologically Correct and Compact Terrain Representation
Site Visit, Feb 2006

W Randolph Franklin, Caroline Westort, Frank Luk, Metin Inanc, Zhongyi Xie
Rensselaer Polytechnic Institute
mail@wrfranklin.org, 703-447-7808
http://wrfranklin.org

Web Site

- http://nga:24nov03@www.ecse.rpi.edu/ Hompages/wrf/Research/geostar-reports/
Team Summary

- Prof Randolph Franklin – helping everyone
- Prof Caroline Westort – strategizing about scooping
- Prof Frank Luk – numerical analysis advice
- Metin Inanc – scooping etc
- Zhongyi Xie – ODETLAP
- Dan Tracy – SVD approximations, multiobserver siting
- Duane Niemeyer, and crew, ESRI

Goals of this project

- Represent terrain (terrestrial elevation)
- Compact; lossily
- Bias representations towards legal terrain
- Preserve usefulness for important applications
- Study and perhaps combine several methods
- Implement on the largest datasets.
- Study size / accuracy tradeoffs.
- Uncompression speed is more important than compression speed.
Anticipated Deliverable

- Multistep compression
- Utilities like missing data fillin

Good compression techniques are multistep

JPEG:
1. Rotate RGB -> YCrCb
2. Discrete cosine transform
3. Low-pass filter
4. Arithmetic encode

Text compression:
1. Run length encoding
2. Burrows-Wheeler transformation
3. Move to front
4. Another run length encoding
5. Arithmetic encode
**Project Outline**

- Develop various techniques independently.
  - TIN (*mature*)
  - Scooping (*still preliminary*)
  - ODETLAP (*middling*)
  - Fast connected components (*mature; needs integrating*)
  - Multiobserver siting (*mature; needs integrating*)
- Then fit them together.
- Search targets of opportunity.
  - Missing data infilling

**Test Data Used**

- USGS 1° DEM: Lake Champlain W, a nice mix of mountains, lowlands, and lake.
- DTED level 2 data from March 2005 kickoff meeting.
- SRTM DTED level 2 data from Jan 2006 CDs.
- We’re standardizing on these since they are nicely mountainous:
  - w119n36.dt2
  - w120n38.dt2
  - w121n45.dt2
  - w112n41.dt2
  - w112n40.dt2
**Note on Reported Numbers**

- We report numbers from various experiments.
- Different experiments would have produced different numbers.
- Nevertheless, we believe ours have enough info content to guide us.
- Since we’re often using Matlab scripts, eventually we can test our proposed solution on all the available data.

**TIN Status**

- We can process 10800x10800 arrays of posts in ½ hr on PC w/o external storage
- Dataset formed by catenating 9 3601x3601 cells from data from the Savannah March kickoff meeting.
- Elevation range: 3600.

<table>
<thead>
<tr>
<th>Elevation accuracy</th>
<th>N output triangles</th>
<th>N out pts / N in pts</th>
<th>Exec time, CPU secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>157,735</td>
<td>0.0007</td>
<td>1292</td>
</tr>
<tr>
<td>10</td>
<td>5,320,089</td>
<td>0.023</td>
<td>1638</td>
</tr>
</tbody>
</table>
TIN ctd

- “Feature” points on peaks and ridgelines may be more important.
- Our TIN program selects them automatically; no need for manual identification.
- The points selected for the triangulation are assumed to be important, and will be fed into other methods, like ODETLAP.
- Note that the TIN is a piecewise linear triangular spline. Preliminary experiments with a higher degree spline showed no consistent improvement, and so were discontinued.
- More extensive experiments might be justified.

Scooping Status

Several subprojects:

- 3-axis milling machine experiments with set of simple drills.
- Complete cover test with parameterized sloped drills.
- Theoretical thinking about how scooping is different from, e.g., wavelets.
Our Scooping Strategy is Dynamic

- Hypothesize.
- Experiment.
- Observe.
- Modify hypothesis.
- Repeat.

Drilling Test on Lk Champlain W - Ideas

- Purpose: to learn what types of drills are more useful.
- Matlab program.
- Greedy optimization: at each step, select the drill instance:

$$(\text{drill type}, x, y, z)$$

that removes the most material.
- Drill instances will generally overlap.
- If two drills drill the same post, then the min elevation is used.
- This nonlinearity is radically different from, e.g., wavelets.
**Subtleties of the Greedy Optimization**

- The drill that removes the most material at the start may be rendered completely redundant by later drills.
- Reminiscent of stepwise multiple linear regression – an initially important variable may ultimately be unimportant.
- Future possibility: removing as well as adding drills.

---

**Ideas for Encoding the Drills**

- Note that the order of drill application is immaterial.
- Therefore group all the instances of the same drill together, to save space.
- Then sort the instances by location and delta encode the locations, since successive instances are relatively close.
**Drilling Test on Lk Champlain W - Observations**

- 1201^2 posts.
- Initial volume of extra material: 1.9E9
- Remaining after 10K steps: 4.2E7.
- Frequency of use of each drill type →
- Surprising observation: flat bottom drills beat curved bottoms.

<table>
<thead>
<tr>
<th>Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square, D=1</td>
<td>0</td>
</tr>
<tr>
<td>Square, D=3</td>
<td>35</td>
</tr>
<tr>
<td>Circle, D=5</td>
<td>2</td>
</tr>
<tr>
<td>Circle, D=10</td>
<td>129</td>
</tr>
<tr>
<td>Circle, D=20</td>
<td>9634</td>
</tr>
<tr>
<td>Parabola, size 5</td>
<td>0</td>
</tr>
<tr>
<td>Parabola, size 10</td>
<td>0</td>
</tr>
<tr>
<td>Parabola, size 20</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 5</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 10</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 20</td>
<td>0</td>
</tr>
</tbody>
</table>

**More General (Sloped) Drills**

- As in any approximation scheme, there is a tradeoff between the power of each basis element, and the number needed.
- However, more powerful basis elements need more bits to encode.
- IMHO the sweet point occurs when the basis elements start to resemble the object being approximated.
- Purpose: to better understand scooping, while initiating experiments in slope-preservation during lossy compression.
- Underlying assumption: little long range correlation of elevation or slope.
Sloped Drill Details

- Try square drills with sloped flat faces.
- For simplicity, the drill instances don’t overlap.
- 2 more degrees of freedom per drill instance.
- Method:
  - Cover the cell with a regular tiling of drills of size 5.
  - Each drill face’s slope is adjusted with SVD to best fit the data.
- Test data:
  - USGS Lake Champlain W cell.
  - SRTM w124 nt38.dt2

Regular Tiling Pros and Cons

- Tiles may not end where they should, e.g., ridges, valley bottoms are all good places for a tile to end and a new tile to start.
- Tiles may end where they should not, e.g., flat areas, water bodies are places where a tile should extend as much as possible.
- Regular tiling may create thin slivers toward the end of the data, which cannot be modeled with planar tiles.
- Regular tiling imposes a hard limit on compression achievable using the method. The tile size limits the number of points which can be grouped together.
- Regular tiles cannot alleviate these problems using any method of shifting or combination of different regular tiling strategies.
- Regular tiling proves a point though. It shows that terrain can be approximated locally using planar tiles.
- This suggests trying image segmentation.
Sloped Drill on Lake Champlain

- Test data: USGS Lake Champlain W cell. (Data range: 1576).
- Over 50% of elevation posts are exactly represented.
- Max error: 101, but that was rare.
- Input data size: 1442401 numbers.
- Output data size: 172800 numbers.
- Drill size: 5.
Error Statistics

Sloped Drill on SRTM Data

- W124N38 dataset.
- Elevation range: -12 to 1022.
- 3601x3601 elevation postings, ¾ ocean.
- 99.97% of the postings are within 16m of the original.
Sloped Drill on w124n38 SRTM Data

W111° N31° Slope Drilling

- Plot shows original data
- 3601x3601
- Nice elevation range
100x100 Detail from W 111° N 31°
Tile Size = 3

100x100 Detail from W 111° N 31°
Tile Size = 5
100x100 Detail from W 111° N 31°
Tile Size = 7

Percent of Elevation Errors on
W 111° N 31°
ODETLAP Status

- This is our overdetermined Laplacian PDE.
- Processing 400x400 arrays of elevation posts.
- Initially sample that with a subarray of regularly spaced points, every K points in each direction.
- When computing a complete surface from the sample points, parameter R trades off accuracy vs smoothness.
- Observe tradeoff of data size versus K, R on a mountainous region of the USGS Lake Champlain W level 1 DEM.
- Next step: use outstanding points from TIN, which should be better than regular points.

Lk Champlain ODEPLAP Experiments

<table>
<thead>
<tr>
<th>K</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>30</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
<td>0.5</td>
<td>2.0</td>
<td>6.1</td>
<td>17</td>
<td>0.4</td>
<td>1.9</td>
<td>7.6</td>
<td>19</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0.5</td>
<td>2.0</td>
<td>6.1</td>
<td>17</td>
<td>0.9</td>
<td>4.4</td>
<td>15</td>
<td>30</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.6</td>
<td>7.5</td>
<td>21</td>
<td>40</td>
<td>65</td>
<td>2.4</td>
<td>11</td>
<td>28</td>
<td>49</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.1</td>
<td>17</td>
<td>39</td>
<td>63</td>
<td>92</td>
<td>6.5</td>
<td>25</td>
<td>52</td>
<td>79</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>36</td>
<td>70</td>
<td>100</td>
<td>137</td>
<td>12</td>
<td>43</td>
<td>81</td>
<td>115</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>50</td>
<td>92</td>
<td>134</td>
<td>164</td>
<td>16</td>
<td>61</td>
<td>122</td>
<td>158</td>
<td>179</td>
<td></td>
</tr>
</tbody>
</table>

R: smoothness vs accuracy
K: spacing of fitted points
**ODETLAP on Bigger Datasets**

- John Childs masters, *A Two-Level Iterative Computational Method for Solution of the Franklin Approximation Algorithm for the Interpolation of Large Contour Line Data Sets*
- Uses a Paige-Saunders conjugate gradient solver with a Laplacian central difference approximation solver for the initial estimate
- However, we'll use our Matlab solver for the moment since it's easier.

---

**Using Important Points**

- The preceding fit a surface to a regular grid of points.
- Fitting “important” points should be better.
- Use our TIN program.
- At each step, TIN inserts the point farthest from the existing surface.
- Use the first N points selected by TIN.
**ODETLAPping TIN Points**

- Test: 400x400 sections: w111n3110, 3111, 3112.
- Compare ODETLAPping first 1000 points selected by TIN with regular grid of 1000 points.
- Measure average, max. abs error over all 40,000 original points.
- TIN: average is worse but max is better, but up to factor of 5.
- I.e., TIN points produce a better conditioned surface.
- Table is an extract of the data, for R=0.3

<table>
<thead>
<tr>
<th>Data</th>
<th>TIN average error</th>
<th>Regular average error</th>
<th>Refined average error</th>
<th>TIN maximum error</th>
<th>Regular maximum error</th>
<th>Refined maximum error</th>
</tr>
</thead>
<tbody>
<tr>
<td>w111n3110</td>
<td>5.6</td>
<td>3.4</td>
<td>5.3</td>
<td>30.7</td>
<td>98.1</td>
<td>27.0</td>
</tr>
<tr>
<td>w111n3111</td>
<td>10.6</td>
<td>9.0</td>
<td>10.4</td>
<td>112.6</td>
<td>133.7</td>
<td>66.2</td>
</tr>
<tr>
<td>w111n3112</td>
<td>2.7</td>
<td>1.7</td>
<td>2.6</td>
<td>20.4</td>
<td>114.0</td>
<td>15.6</td>
</tr>
</tbody>
</table>

**Fitting Regular vs TIN Points**

- Original W111n3110 data
- Fitting TIN points matches the character of the surface better

2/16/2009 9:05 PM
**Fitting Regular vs TIN Points**

- Original W111n3111 data
- Fitting TIN points matches the character of the surface better

**Iterating ODETLAP**

- Run ODETLAP on a grid or TIN of points.
- Find the points farthest from the fitted surface.
- Insert them into the set of points to be fitted.
- Rerun ODETLAP.
- Max error reduced.
- See previous table.
**Smoothing With ODETLAP**

- 400x400 piece of Lk Champlain
- Fitting all points
- Increasing smoothing from R=0.1 to R=1.
- Average abs error increased from .1 to 2.

---

**Filling in Missing Data**

- ODETLAP can be used to fill in missing holes, although this isn’t its main purpose.
- We extensively tested it on the Lake Champlain W cell:
  - 20 different locations
  - Circles of missing data of radius 5, 10, 20, 40
  - Emphasize either accuracy or smoothness.
  - Plot contours of the real data and of the interpolated data
  - Histogram the error
- Next slide shows one experiment.
- Note that, even though the (fine) contours appear different, the average error may be small.
**Smoothing Data**

- ODETLAP could be used to smooth artifacts such as the visible contour lines below.
- Indeed, preventing these was always a design goal of ODETLAP.
**Multiobserver Siting Status**

- SW written by WRF, and extended Christian Vogt as a masters thesis.
- It works, but complicated and messy.
- RPI and ESRI have agreed on general contract terms.
- Dan Tracy is packaging it so we can use it to evaluate our compression.
- Then ESRI will productize it in ARC-GIS.

**With or w/o intervisibility**

[Images of maps showing intervisibility comparisons]

---

Feb 2006 site visit
Multiobserver sitting ctd

- Trying to delay using it for evaluation.
- More important to study compression techniques w/o diversions.
- Easier to measure RMS error than sitting accuracy
- E.g., can measure RMS error easily and quickly in Matlab, while sitting accuracy is slower and would require callout to C++.
- Assume smaller RMS error => better siting.

Hydrology Status

- Can compute workflow ignoring the problem of local minima (but that’s easy).
- However, local watersheds trap all the long rivers.
- Fast connected component program works.
- To do: use the latter to identify local watersheds so they can be removed.
- Not currently working on this theme, since ArcGIS is good enough for the moment.
Publicity

- RPI announced our award in a press release.
- 4 blogs picked it up:
  - ACM Technews.
  - Defencetalk
  - Surfwax
  - Zdnet
- WRF interviewed on WGY radio on 11/3/05.
- Current Rensselaer alumni magazine has ½ page on this project.

Internal Management

- Semiweekly meetings.
- Internal web site, being converted to a wiki.
- Web site, to be converted to wiki, to communicate results to DARPA, NGA.
Budget Changes From Proposal

- Less fancy equipment. We are buying a 64GB memory quad Opteron to process large datasets.
- Smaller subaward to ESRI.
- More people, both faculty and students.
- Faculty to bring a broader range of expertise to the project.
- Students to become a future valued human resource.

Internal Outreach

- Frank Luk will teach a grad course next fall on, tentatively, *Numerical Methods for Landscaping*.
- Caroline Westort has been asked by Civil Engineering to teach a course.
Things We Could Do If You Want

- (We have more ideas than resources.)
- Robust multiobserver siting – site observers so each target has 2 observers.
- Site robustly against data errors.
- Site observers so hidden regions are disconnected pieces.
- Find moving observer trajectories for:
  - Tourists
  - Smugglers

Future Strategy

Our ultimate solution is expected to be a sequence of all of our techniques:

- Use TIN to find important points.
- Use them to guide scooping.
- Smooth the resulting surface with ODETLAP.
- Remove any local minima with hydrology programs.
Compact Visibility-Preserving Terrain Representations
Review, April 2006

Prof W Randolph Franklin, Prof Caroline Westort, Prof Frank Luk, Prof Barbara Cutler, Metin Inanc, Zhongyi Xie, Dan Tracy
Rensselaer Polytechnic Institute
mail@wrfranklin.org, 703-447-7808
http://wrfranklin.org

Goals of this project

- Alternate terrain representations
- Compact; lossy - size / quality tradeoffs.
- Bias representations towards legal terrain
- Process datasets up to 10000x10000
- Uncompression speed is more important than compression speed.
- Evaluate on visibility, mobility metrics.
### Team
- Prof Randolph Franklin – helping everyone
- Prof Caroline Westort – strategizing about scooping
- Prof Frank Luk – numerical analysis advice
- Prof Barbara Cutler – computer graphics advice
- Metin Inanc – scooping etc
- Zhongyi Xie – ODETLAP
- Dan Tracy – SVD approximations, multiobserver siting
- Duane Niemeyer and crew, ESRI – implementing siting toolkit
- Note: COL Clark Ray has retired from USMA.

### Accomplishments and Status
- Proceeding with researching alternative, more compact, representations of elevation data.
- Now combining the various representations for greater power.
- Initial evaluations under the observer siting and visibility metric
- Starting to employ the mobility metric.
Sample Results

- TIN: represented a 10800x10800 array to 3% max elevation error with 157,735 triangles.
- Scooping: represented w111n31 with 7x7 linear scoops with average error 0.1% and max error 2%.
- Using 7x7 scoops on one 3592x3592 dataset, multiobserver siting had only 6.5% error.
- ODETLAPping 400x400 piece of Lake Champlain W with 1/9 the points: error was 0.9m (0.1%).
- Combining TIN with ODETLAP: captures essence of surface with very few points.
- ODETLAP: Can fill radius 40 circles of missing data.

Test Data Used

- USGS 1° DEM: Lake Champlain W, a nice mix of mountains, lowlands, and lake.
- DTED level 2 data from March 2005 kickoff meeting.
- SRTM DTED level 2 data from Jan 2006 CDs.
- We’re standardizing on these nicely mountainous cells:
  * w119n36.dt2, w120n38.dt2, w121n45.dt2, w112n41.dt2, w112n40.dt2
- Per NGA request in Feb, we added some less mountainous cells:
  * w112n32.dt2, w113n32.dt2, w113n33.dt2, w113n40.dt2, w113n41.dt2, w114n32.dt2, w114n33.dt2, w115n33.dt2, w119n33.dt2
Test Data Complexities

Varying Resolution

Bunched Elevations
Testing Protocol

- Compute some property on the original terrain representation (large matrix of elevation posts), and again on the alternative representation.
- Measure the difference.
- Sample various test datasets.
- Tradeoff size vs quality.

Protocol 1: Elevation Testing

1. Measure both average absolute error and RMS error
   - Some representations do better at one than other.
   - Examine terrain to see if features captured.
     - Important but hard to quantify.
Protocols 2-4: Visibility Testing

Various levels of complexity:
2. Evaluate differences in observer viewsheds
3. Evaluate visibility index of every observer.
4. Evaluate multiobserver siting quality.

Protocol 2: Viewshed Testing

- Select parameters:
  - Radius of interest (R)
  - Observer, target heights (H)
- Compute viewshed bitmaps of many observers on both the original and the alternative representations. *(we can do this fast for large R).*
- For some observer O, let $V_o$ be its viewshed on the original, and $V_a$ be its viewshed computed on the alternative representation. $V_o$ and $V_a$ are bitmaps (i.e., sets of target points visible by O). Report $|V_o - V_a| + |V_a - V_o|$. 
Protocol 3: Visibility Index Testing

- Consider each post in term as an observer.
- Compute its visibility index.
  - Monte Carlo sampling: pick T random targets, compute their visibility, and report the fraction visible.
- Produce a map of all the visibility indexes.
- Compare the visibility index map of the original terrain representation to the map of the alternative representation.

Protocol 4: Multiple Observer Siting Testing

- Site a set of observers, \( S_o \), on the original terrain rep.
- Site a set of observers, \( S_a \), on the alternative terrain rep.
- Transfer \( S_a \) to the original rep.
- Compare quality of \( S_a \) to \( S_o \).
Multiobserver Siting Status

- Software written by WRF, and extended Christian Vogt as a masters thesis.
- It works, but is complicated and messy.
- ESRI is productizing it with ArcGIS.
- Dan Tracy is using it internally to evaluate our compression.

With or w/o intervisibility

<table>
<thead>
<tr>
<th>Intervisibility enforced</th>
<th>No intervisibility required</th>
</tr>
</thead>
</table>

Color -> elevation; Black -> hidden.
**Alternative Representations**

1. TIN
2. Scooping
3. ODETLAP
4. Combinations of the above, e.g., ODETLAP uses TIN points.

---

**Note: Good compression techniques are multistep**

JPEG:
1. Rotate RGB -> YCrCb
2. Discrete cosine transform
3. Low-pass filter
4. Arithmetic encode

Text compression:
1. Run length encoding
2. Burrows-Wheeler transformation
3. Move to front
4. Another run length encoding
5. Arithmetic encode
TIN Status

- We can process 10800x10800 arrays of posts in ½ hr on PC.
- No external storage is used.
- Dataset formed by catenating nine 3601x3601 cells from data from the Savannah March kickoff meeting.
- Elevation range: 3600.

<table>
<thead>
<tr>
<th>Max elevation error</th>
<th>N output triangles</th>
<th>N out pts / N in pts</th>
<th>Exec time, CPU secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>157,735</td>
<td>0.0007</td>
<td>1292</td>
</tr>
<tr>
<td>10</td>
<td>5,320,089</td>
<td>0.023</td>
<td>1638</td>
</tr>
</tbody>
</table>

TIN Features

- Progressive resolution since points are inserted greedily.
- “Feature” points on peaks and ridgelines, and edges joining them, may be more important.
- Our TIN program selects them automatically; no need for manual identification and constrained triangulation.
- The points selected for the triangulation are assumed to be important, and can be fed into other methods, like ODETLAP.
- TIN is a piecewise linear triangular spline. Preliminary experiments with a higher degree spline showed no consistent improvement, and so were suspended.
Lossless TIN

- TIN is inherently lossy.
- Make it lossless:
  - Use DETLAP to fit a surface to the TIN points.
  - Compute the errors.
  - Compress (x, y), z, errors each with PAQ3N.
- Test on w11n3111, 400x400.

Coding the TIN Representation

- Q: How to code the \{(x, y, z)\} TIN points?
- Note: order of the set elements is immaterial.
- A1: Use bitmap coding techniques for \{(x, y)\}.
- There is an info-theoretic limit here.
- Order of z is now determined.
- A2: Compress z with a sequence of Burrow Wheeler, Run Length and Arithmetic Coding.
Alternative Representations

1. TIN
2. Scooping
3. ODETLAP
4. Combinations of the above, e.g., ODETLAP uses TIN points.

Scooping Representations

- This is longterm research.
- The goal is to smash through the information theoretic barrier to terrain compression by utilizing geologic information.
- We are pursuing several representations in parallel.
**Scooping Status**

Several subprojects:

- 3-axis milling machine experiments with set of simple drills.
- Complete cover test with parameterized sloped drills.
- Theoretical thinking about how scooping is different from, e.g., wavelets.

**Dynamic Strategy**

- Hypothesize.
- Experiment.
- Observe.
- Modify hypothesis.
- Repeat.
Drilling Test on Lk Champlain W - Ideas

- Purpose: to learn what types of drills are more useful.
- Matlab program.
- Greedy optimization: at each step, select the drill instance:

\[(\text{drill type, } x, y, z)\]

that removes the most material.
- Drill instances will generally overlap.
- If two drills drill the same post, then the min elevation is used.
- This nonlinearity is radically different from wavelets.

Subtleties of the Greedy Optimization

- The drill that removes the most material at the start may be rendered completely redundant by later drills.
- Reminiscent of stepwise multiple linear regression – an initially important variable may ultimately be unimportant.
- Future possibility: removing as well as adding drills.
Encoding the Drills (Future)

- Note that the order of drill application is immaterial.
- Therefore, when coding, group all the instances of the same drill together, to save space.
- Then sort the instances by location and delta encode the locations, since successive instances are relatively close.

Drilling Test on Lk Champlain W - Observations

- 1201^2 posts.
- Initial volume of extra material: 1.9 \times 10^9
- Remaining after 10,000 steps: 4.2 \times 10^7
- Mean error: 30m (2%)
- Frequency of use of each drill type
- Surprising observation: flat bottom drills beat curved bottoms.

<table>
<thead>
<tr>
<th>Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square, D=1</td>
<td>0</td>
</tr>
<tr>
<td>Square, D=3</td>
<td>35</td>
</tr>
<tr>
<td>Circle, D=5</td>
<td>2</td>
</tr>
<tr>
<td>Circle, D=10</td>
<td>129</td>
</tr>
<tr>
<td>Circle, D=20</td>
<td>9834</td>
</tr>
<tr>
<td>Parabola, size 5</td>
<td>0</td>
</tr>
<tr>
<td>Parabola, size 10</td>
<td>0</td>
</tr>
<tr>
<td>Parabola, size 20</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 5</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 10</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 20</td>
<td>0</td>
</tr>
</tbody>
</table>
More General (Sloped) Drills

- Tradeoff powerful, large to encode, basis elements, vs small simple elements, of which we need more.
- Sweet point: basis elements resemble object being approximated.
- Purpose: to better understand scooping, while initiating experiments in slope-preservation during lossy compression.
- Underlying assumption: little long range correlation of elevation or slope.

Regular Scoop Details

- 7x7 Scoop size will represents 49 elevations using only 3 coefficients
- 7 is not a magic number but good enough for Level 2 DTED cells
- Large Errors are rare and mean error is very low, less than 2m
- Each scoop is a tilted plane which minimizes the error
- Regularity brings simplicity to the representation
Regular 7x7 Tile Scoop Representation

W111°N31° Reconstructed (Left), Error (Right)
Factor of 49 reduction in number of points

Percent of Elevation Errors on W 111° N 31°
7x7 Regular Sloped Scoop VIX Evaluation

- Comparing Postings with Visibility Index Larger Than 80%
- Original (Above), Reconstructed (Below)
- Yellow: High VIX
- Green: Low VIX
- Difference is not easy to discern

Viewshed Evaluation of Regular Scooping

- Dataset: 3595x3595
- Number of observers: 81
- Elevation range: 809 to 2882.
- Observer/target height is 10.
- Radius of interest: 300.

<table>
<thead>
<tr>
<th>Representation</th>
<th>%Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>tile3</td>
<td>1.31</td>
</tr>
<tr>
<td>tile5</td>
<td>3.67</td>
</tr>
<tr>
<td>tile7</td>
<td>6.52</td>
</tr>
<tr>
<td>tile75</td>
<td>5.62</td>
</tr>
<tr>
<td>tileavg75</td>
<td>3.16</td>
</tr>
</tbody>
</table>
Multiobserver Viewshed Comparison

Original Terrain Rep    Tile 7 Alternate Terrain Rep

Alternative Representations

1. TIN
2. Scooping
3. ODETLAP
4. Combinations of the above, e.g., ODETLAP uses TIN points.
ODETLAP Review

- Solve an overdetermined variant of a Laplacian PDE.
  - Known pts: \( z_{ij} = h_{ij} \)
  - All pts: \( 4z_{ij} = z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1} \)
- Easily processes 400x400 arrays of elevation posts in Matlab.

ODETLAP Advantages

- Infers local maxima.
- Surface doesn’t droop.
- Utilizes isolated data, if available.
- Interpolates broken contours.
- Conformal (handles nested kidney-bean contours)
- Conflates inconsistent data, with user-defined weights.
How Can ODETLAP Infer Maxima?

- *(This question was raised at San Diego.)*
- Points interpolated with a Laplacian PDE must fall inside the range of boundary value points.
- We’re not doing a Laplacian PDE, but an extension of it.
- We can generate values outside the range of boundary value points.

Four Matlab Interpolation Techniques on Nested Square Contours

- Difficult interpolation example
- Irregular silhouette edges and flat top illustrate these methods' limitations.
**ODETLAP on Nested Squares**

- Various smoothness settings are possible.
- R=3 gives
  - Completely smooth silhouettes,
  - Average error = 2.7%
  - Max error = 12%.

**ODETLAP on Bigger Datasets**

- John Childs masters, *A Two-Level Iterative Computational Method for Solution of the Franklin Approximation Algorithm for the Interpolation of Large Contour Line Data Sets*
- Uses a Paige-Saunders conjugate gradient solver with a Laplacian central difference approximation solver for the initial estimate
- However, we'll use our Matlab solver for the moment since it's easier.
ODETLAP on Regular Points

- Initially sample that with a subarray of regularly spaced points, every K points in each direction.
- When computing a complete surface from the sample points, parameter R trades off accuracy vs smoothness.
- Observe tradeoff of data size versus K, R on a mountainous region of the USGS Lake Champlain W level 1 DEM.

Lk Champlain ODEPLAP Experiments

<table>
<thead>
<tr>
<th>K</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1 0.5 2.0 6.1 17</td>
</tr>
<tr>
<td>2</td>
<td>0.4 1.9 7.6 19 37</td>
</tr>
<tr>
<td>3</td>
<td>0.9 4.4 15 30 53</td>
</tr>
<tr>
<td>4</td>
<td>1.6 7.5 21 40 65</td>
</tr>
<tr>
<td>5</td>
<td>2.4 11 28 49 75</td>
</tr>
<tr>
<td>7</td>
<td>4.1 17 39 63 92</td>
</tr>
<tr>
<td>10</td>
<td>6.5 25 52 79 112</td>
</tr>
<tr>
<td>15</td>
<td>10 36 70 100 137</td>
</tr>
<tr>
<td>20</td>
<td>12 43 81 115 153</td>
</tr>
<tr>
<td>30</td>
<td>14 50 92 134 164</td>
</tr>
<tr>
<td>50</td>
<td>16 61 122 158 179</td>
</tr>
</tbody>
</table>

K: spacing of fitted points  
R: smoothness vs accuracy  
(Data range: 1378)
Factor of 9 Reduction

- ODETLAP used 1/9 as many points to represent this surface with an elevation error of 0.9m in a range of 1378.

Alternative Representations

1. TIN
2. Scooping
3. ODETLAP
4. Combinations of the above, e.g., ODETLAP uses TIN points.
**ODETLAPping Important Points**

- The preceding fits a surface to a regular grid of points.
- Fitting “important” points should be better.
- Use our TIN program, which, at each step, inserts the point farthest from the existing surface.
- Use the first N points selected by TIN.

---

**ODETLAPping TIN Points**

- Test: 400x400 sections: w11n3110, 3111, 3112. R=0.3
- Compare ODETLAPping first 1000 points selected by TIN with regular grid of 1000 points.
- Measure average, max, abs error over all original points.
- TIN: average is worse but max is better, but up to factor of 5.
- TIN points produce a better conditioned surface.
- Refined: Insert worst points into TIN ODETLAP. Result: even better conditioned surface.

<table>
<thead>
<tr>
<th>Data</th>
<th>TIN average error</th>
<th>Regular average error</th>
<th>TIN maximu m error</th>
<th>Regular maximu m error</th>
</tr>
</thead>
<tbody>
<tr>
<td>w11n3110</td>
<td>5.6</td>
<td>3.4</td>
<td>5.3</td>
<td>30.7</td>
</tr>
<tr>
<td>w11n3111</td>
<td>10.6</td>
<td>9.0</td>
<td>10.4</td>
<td>112.6</td>
</tr>
<tr>
<td>w11n3112</td>
<td>2.7</td>
<td>1.7</td>
<td>2.6</td>
<td>20.4</td>
</tr>
</tbody>
</table>
Fitting Regular vs TIN Points

- Original W111n3110 data (160,000 points)
- Fitting TIN points matches the character of the surface better

Fitting 100 Regular Points
Fitting 100 TIN Points

Fitting Regular vs TIN Points

- Original W111n3111 data
- Fitting TIN points matches the character of the surface better

Fitting 36 Regular Points
Fitting 30 TIN Points
**W113n3311 - Regular vs Refined TIN**

- Original Dataset – 160K Pts
- 10000 Regular Points
- 3000 TIN Points Plus 100 Worst Points

**Filling in Missing Data**

- ODETLAP can be used to fill in data holes, although this isn’t its purpose.
- In San Diego, we were asked to study this more.
- We extensively tested it on the Lake Champlain W cell:
  - 20 different locations
  - Circles of missing data of radius 5, 10, 20, 40
  - Emphasize either accuracy or smoothness.
  - Plot contours of the real data and of the interpolated data
  - Histogram the error
- Results on an interactive web site.
- Next slide shows one experiment.
- The average error is small.
Publicity

- RPI announced our award in a press release, which was preapproved by DARPA.
- 4 blogs picked it up:
  - ACM Technews
  - Defencetalk
  - Surf wax
  - Zdnet
- WRF interviewed on WGY radio on 11/3/05.
- Current Rensselaer alumni magazine has ½ page on this project.
Internal Management

- Semiweekly meetings.
- Internal web site, being converted to a wiki.
- Web site, to be converted to wiki, to communicate results to DARPA, NGA.

Next Steps – Near Term Goals

- More detailed evaluation using mobility.
- Find moving observer trajectories for:
  - Tourists
  - Smugglers
- Path planning to remain in the viewsheds, or in the viewsheds’ complements.
- Optimizing.
- More scooping research.
Path Planning Proposed Process 1

- On a compressed alternate representation, compute a smuggler’s path, never entering any observer’s viewshed.
- Evaluate that path on the original noncompressed rep.
- Report the fraction of the path that is visible, as a function of the data size.
  * Lower is better.

Path Planning Proposed Process 2

- On a compressed alternate representation, compute a smuggler’s path, with minimum total visibility.
- Evaluate that path on the original noncompressed rep.
- Report the path’s correct total visibility as a function of the data size.
  * Lower is better.
Flythru of Tile 7 Representation

- The terrain looks good.

Next Steps – Midterm Goals

- Frank Luk will teach a grad course next fall on, tentatively, *Numerical Methods for Landscaping*.
- Caroline Westort may teach a terraforming course for Civil Engineering.
- We’re studying forming a consortium on digital terrain modeling and construction applications.
**Next Steps – Long Term Goals**

*(depending on interest)*

- Robust multiobserver siting – site observers so each target has 2 observers.
- Site robustly against data errors.
- Site observers so hidden regions are disconnected pieces.
- More path planning optimization.
1. Recent significant accomplishments

1. Since Snoqualmie, we have a preliminary path-planning program running. It plots a path between a user-specified source and goal point while avoiding the viewsheds of all the observers. An example output is shown on the **single summary slide** prepared in April.

2. The more we use ODETLAP, our overdetermined Laplacian PDE method, the more we like it. As shown in Snoqualmie, it can fit a recognizable surface to even very few points. (We showed one example where a surface fit to 36 points had some major features recognizable.) ODETLAP can also fill in missing data. It could also determine a regular grid of elevation posts from an irregular set of input data.

3. Preliminary experiments indicate that multiple observer siting is rather
robust with respect to alternative compressed terrain representations. In more detail:

a. On a compact terrain representation, site a the best set of observers we can, that is find a set of observers whose joint viewshed is as large as possible.

b. Do the same on the original terrain representation.

c. Compute (on the original terrain) the area covered (i.e., the joint visibility index) of both sets of observers.

d. Note how the joint visibility index of the observers from the compressed representation is often not much smaller than the joint visibility index of the observers from the original representation.

e. Therefore we hope to be able to compress terrain quite aggressively w/o hurting the observer siting much.

4. One preliminary experiment, shown in last month's summary slide, indicated that a shortest path avoiding viewsheds computed on a compressed representation is largely still hidden when evaluated on the original representation,

2. Obstacles

1. ODETLAP, as we have implemented it (in Matlab), is processing only 400x400 arrays. Processing much larger arrays would require either specialized techniques, or leaving Matlab altogether. John Childs, a former masters student of Franklin, (before this contract) processed larger arrays with more sophisticated techniques.

2. It was harder than anticipated to locate useful path-planning implementations. An extensive web search found not much that was really useful, so we mostly have been implementing things ourselves.

3. We are still using relatively simple scoops in the scooping representation. They do compress surprisingly well, as shown on the summary slide. However, using geological information in the scooping
process is complicated.

3. Next steps

1. We are continuing on the various parallel paths mentioned at Snoqualmie.

2. Since JPEG was mentioned as relevant, we are experimenting with using it to encode the coefficients produced by any of our other methods. We anticipate that the resulting size dataset will be smaller than if JPEG is used to encode the raw elevation matrices.

3. We are working path planning into the project in a more systematic way. My summary slide last month was a one-time hack in order to get the requested results in time. E.g. the path source and goal were hard coded into the planning program. We will clean this up to make it more generally useful.

4. Various people's tasks

1. Metin Inanc is writing a terrain assistant to help the user to partition a cell into regions of similar characteristics so that different regions can be encoded differently.

2. Zhongyi Xie is exploring ODETLAP's possibilities. One avenue is to incrementally insert the worst point into the set of points being fitted and repeat. He already does one iteration of this, inserting 100 worst points, after fitting 1000 TIN points. We want to see how far this can be taken.

3. Dan Tracy is working on siting and mobility.

4. Caroline Westort is investigating other scooping or sculpting techniques. Steve Martin, a junior, will be helping her this summer.

5. Frank Luk is researching formal mathematical foundations for these ideas.

6. Marcus Alvim Andrade, a faculty member from Brazil, visited me last fall.
for a few weeks. He is interested in a continuing collaboration, and in visiting RPI for a postdoc next year, paid by the Brazilian government. He will be an asset to this project; we are thinking where the best fit will be.

7. Next fall, two new grad students supported on teaching assistantships are expected to join. My (WRF) major management issue will be to carve out separate projects so that they are not all stepping on each others' toes, while retaining flexibility to change depending on experimental observations.

5. Project Web Site

We have a project web site, http://wrfranklin.org/pmwiki/GeoStar, to contain all our reports. The password is donsawowl.

6. Budget Spendout

Per RPI's online accounting system, the contract has spent $320,138.21. That does not include a large amount of commitments for several people's summer salary. We are on track.
Smugglers’ Path Planning on 16x Compressed “Scooped” Terrain Representation

- Original 3595x3595
  - W11N31 Terrain: 12,924,025 d.f., Elev Range=2071

- Compressed (7x7 Scoop): 791,267 d.f. (16x reduction), Mean abs error=1.7 (0.1%).

- Compressed: Shortest Smugglers Path Computed Avoiding All 324 Viewsheds of Optimally Sited Observers

- Original: Joint Viewshed Computed for Same 324 Observers

- Evaluation: Optimal Path from Compressed Terrain Tested on Original Terrain Viewsheds – 14 of 4767 Points (0.3%) Are Erroneously Visible

RPI Geo* Final Report
May 2006 Slide
Summary Information
- Professional Personnel
- Project Goals
- Summary of Accomplishments and Status
- Test Datasets and Protocols
- Alternative Terrain Representations
  - TIN
  - Features
  - Improvements This Year
  - Preliminary Testbed
  - Status
  - Lossless TIN
- Scooping
  - Degree-0 Scoops
  - Degree-1 Scoop
  - Sculpting
- ODETLAP (Overdetermined Laplacian PDE)
  - Choices for Input Points
- Filling in Missing Data Holes
- Multiobserver Siting
  - Effect of Reducing Resolution
- Smugglers' Path Planning
- Using Smugglers' Path Planning to Evaluate Alternate Representations
- Coding the Representation's Coefficients
- Comparison of Actual Achievements with Goals
- Cost Overruns
- Interactions Between RPI and Sponsors
- Publications
- Press and Blog Mentions
- Technical Notes

http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Geo...
Professional Personnel

Various personnel, ranging from faculty members to grad students and undergrad students, are currently being supported by the project.

1. Dr W. Randolph Franklin, Associate Professor, RPI, PhD (Applied Math, Harvard): Strategizing, managing, writing, prototype coding, "(parttime on this project during the academic year, two months in summer)."

2. Dr Caroline Westort, Research Assistant Professor, RPI, PhD (Geography, Zürich): Strategizing about scooping, "(fulltime from August 2005)."

3. Dr Frank Luk, Professor, RPI, PhD (Computer Science, Cornell): Numerical analysis advice, "(two months this summer)." Prof Luk is a former chair of CS, and is an internationally renowned expert in Scientific Computation.

4. Metin Inanc, Research Assistant and PhD candidate (RPI): Incremental TIN, using ODETLAP to fill in missing data holes, the scooping alternative terrain representation, "(halftime during the academic year, fulltime in the summer)."

5. Zhongyi Xie, Research Assistant and PhD candidate (RPI): Using ODETLAP to reconstruct a surface from the minimal number of points generated by a TIN, "(halftime during the academic year, fulltime in the summer)."

6. Dan Tracy: PhD candidate (RPI): SVD approximations, multiobserver siting, path planning, "(supported as a Teaching Assistant during Spring 2006, supported as a Research Assistant on the contract halftime during the academic year starting Fall 2006, fulltime in the summer starting 2006)."

7. Jared Scheuer, junior: studied aspects of ODETLAP during summer 2005. He measured the accuracy of either subsampling the surface, or of having a region of missing data, and using this technique to fill in the missing data.

8. Rositsa Tancheva, junior: researched non photorealistic rendering (NPR) in computer graphics, in order to apply that to terrain. This was exploratory work that might lead to another new representation.

9. John Shermerhorn, junior: also looking at some terrain issues, such as creating a testbed in which we can try our ideas.
10. Joe Roubal, ESRI: implementing the multiobserver siting toolkit in ArcGIS.


12. COL Clark Ray, PhD, provided valuable advice while at the USMA West Point. However, he has since retired from the Army, and unfortunately is no longer active on this project.

**Project Goals**

These may be summarized as follows.

1. Study alternate terrain representations that are more compact.
2. Since the new representations will therefore be lossy, evaluate the size / quality tradeoffs.
3. These new representations ought to make it easier to represent legal terrain than illegal terrain.
4. We wish to process datasets up to 10000x10000 elevation posts.
5. With these new representations, uncompressed speed is more important than compression speed.
6. The new representations are to be evaluated on metrics such as visibility and mobility.

**Summary of Accomplishments and Status**

We are proceeding with researching alternative, more compact, representations of elevation data. We are now combining the various representations for greater power. Our initial evaluations of our alternative representations use, in addition to RMS elevation error, the observer siting and visibility metric. We are starting to employ the mobility metric. Here are some sample results.

1. Our incremental Triangulated Irregular Network (TIN) program can process a 10800x10800 array of posts in internal memory on a PC. The accuracy is impressive; one dataset uses only 157,735 triangles to represent the 11,664,000 posts to a 3% maximum elevation error.

2. Our scooping representation processed the W111N31 cell with 7x7 linear scoops to an average elevation error of 0.1%, and maximum error of 2%.

3. We represented one 3592x3592 dataset with 7x7 scoops, and then compared the joint viewsheds resulting from multiobserver siting on the original representation and the alternate. The error was only 6.5%.

4. We tested another new representation, ODET LAP, on a 400x400 piece of the Lake Champlain W cell. After selecting every third post in each of X and Y, that is 1/9 of the points, and using ODET LAP to fit a surface to them, the error was only 0.9 meters, or 0.1% of the elevation range.
5. We are combining TIN with ODETLAP, in order to capture the essence of a surface with very few points.

6. We can use ODETLAP to fill in radius 40 circles of missing data, with excellent results.

Test Datasets and Protocols

These experiments used the following test datasets:

1. USGS Lake Champlain W 1° DEM, which has a good mix of mountains, including Mt Marcy (the NYS highpoint), lowlands, and Lake Champlain.

2. DTED level 2 data from the March 2005 kickoff meeting.

3. SRTM DTED level 2 data from CDs supplied in January 2006.

Of that data, we’re standardizing on these mountainous cells: w119n36.dt2, w120n38.dt2, w121n45.dt2, w112n41.dt2, w112n40.dt2. Per an NGA request in February 2006, we added some less mountainous cells: w112n32.dt2, w113n32.dt2, w113n33.dt2, w113n40.dt2, w113n41.dt2, w114n32.dt2, w114n33.dt2, w115n33.dt2, w119n33.dt2.

Our several testing protocols operate as follows.

1. Compute some property on the original terrain representation, which is a large matrix of elevation posts.

2. Compute that property again on the alternative representation, whether TIN, scooping, or ODETLAP.

3. Measure the difference between the two representations.

4. Evaluate various test datasets.

5. Examine the tradeoff between the size of the dataset and its quality.

Some of the various test protocols are the following.

1. Protocol 1: Elevation Testing. Here we measure both average absolute error and RMS error. The reason is that some representations do better at one than at the other. In addition, we examine the terrain to see if its features are captured. This is important but hard to quantify.

2. Protocols 2-4: Visibility Testing. These contain various levels of complexity:

    a. Evaluate differences in observer viewsheds.
b. Evaluate the visibility index of every observer in the cell.

c. Evaluate multiobserver siting quality.

3. Protocol 2: Viewshed Testing, which proceeds as follows.

   a. Select values for the following parameters: Radius of interest, $R$, and observer and target heights, $H$.

   b. Compute the viewshed bitmaps of many observers on both the original and the alternative representations. We can do this fast even for large $R$.

   c. For some observer $O$, let $V_O$ be its viewshed on the original, and $V_a$ be its viewshed computed on the alternative representation. $V_O$ and $V_a$ are bitmaps (i.e., sets of target points visible by $O$). Report the size of the symmetric difference, $|V_O - V_a| + |V_a - V_O|$.

4. Protocol 3: Visibility Index Testing:

   a. Consider each post in term as an observer.

   b. Compute its visibility index.

   c. Using Monte Carlo sampling, pick $T$ random targets, compute their visibility, and report the fraction visible.

   d. Produce a map of all the visibility indexes.

   e. Compare the visibility index map of the original terrain representation to the map of the alternative representation.

5. Protocol 4: Multiple Observer Siting Testing

   a. Site a set of observers, $S_o$, on the original terrain representation.

   b. Site a set of observers, $S_a$, on the alternative terrain representation.

   c. Transfer $S_a$ to the original representation.

   d. Compare the quality of $S_a$ to that of $S_o$.

**Alternative Terrain Representations**

Our alternative terrain representations are as follows.
1. TIN
2. Scooping
3. ODETLAP
4. Combinations of the above, e.g., ODETLAP uses TIN points.

It is important to note that good compression techniques are multistep. For instance, JPEG combines the following sequential steps:

1. Rotate from the RGB colorspace to the YCrCb colorspace.
2. Perform a discrete cosine transform.
3. Perform a low-pass filter.
4. Arithmetic encode the remaining coefficients.

Text compression, e.g. bzip2, performs the following steps in sequence.

1. Run length encoding.
2. Burrows-Wheeler transformation.
3. Move to front.
4. Another run length encoding.
5. Arithmetic encoding.

Our ultimate recommendation may well be a combination of our various techniques, perhaps as follows:

1. Reconstruct an approximate surface with scooping, or
2. Select significant points with TIN.
3. Smooth the surface with ODETLAP.
4. Fill in any local minima with our fast connected component program. (This program designed to process 3D cubes of bits, up to 600x600x600 voxels, resulting from thresholding a CAT scan. However, it can process 2D datasets as a special case. For example, one 18573x19110 image took 25 CPU seconds to process on a 1600 MHz laptop.)

Note that the last two steps do not increase the information content of our representation. That is, if they are a fixed part of our reconstruction engine, having them operate adds zero bits to the representation of the terrain.

Some experiments were performed by Tracy to compress the terrain using the singular value decomposition (SVD.) Various methods of partitioning the terrain and performing an SVD on each partition were tested. The results were inferior to those of competing compression methods. However, the SVD may still be useful in combination with other compression methods, such as scooping.

TIN
Features

Our version is a progressive resolution method.

1. The initial TIN is just two triangles partitioning the cell.

2. At each step, the post whose elevation is vertically farthest from the triangle containing its projection onto the XY plane is inserted into the triangulation, splitting one triangle into three.

3. Often that point lies on an edge of the triangulation. In that case, one triangle is degenerate.

4. If the new point lies on the cell's border, then its triangle is split into only two triangles.

5. If necessary, a few edges are flipped to maintain the triangulation as Delauney. If one of the new triangles is degenerate, then one of its edges will always be flipped, removing the degeneracy.

This progressive resolution property is not common to some other triangulation methods, which may process the whole set of points at once. Our method may be stopped when any precondition is met. Typical preconditions include the insertion of a predetermined number of points, or the attainment of a predetermined maximum absolute error.

The points that have been inserted at that time are, in some sense, the most important points in the surface. They may then be used by a successive process such as ODETLAP. Since this point may not be widely appreciated, Figure 1 shows an example constructed to be difficult. The test dataset contains two overlapping circular ridges with sharp tops, transected by a railroad embankment with vertical sides.

![Railroad embankment](difficult_test_dataset_for_tin.png)
Figure 2 shows the result of applying TIN until the maximum error is sufficiently small. Note how points are selected along the ridges and embankment sides, and how the edges align with these features and do not cross them. Figure 3 is an enlarged detail.
The data structures used by our TIN method were the result of considerable thought before the first line of code was written. They are very compact, which is why we can process such large cells on a PC. For example, edges are not stored explicitly, but rather, each triangle has pointers to its (one to three) neighboring triangles. That saves space, at the cost of slowing down edge flips.

Feature points on the peaks and ridgelines, and the edges joining them, may be more important properties of the surface. Our TIN program selects them automatically; there is no need for manual identification and constrained triangulation. The points selected for the triangulation are assumed to be important, and can be fed into other methods, like ODETLAP. TIN is a piecewise linear triangular spline. Preliminary experiments with a higher degree spline showed no consistent improvement, and so were suspended.

**Improvements This Year**

Inanc modularized the TIN program and changed it in several ways. The error handling was changed to use C++’s exception handling mechanism. The code in the `main()` was removed and a new function was created to include the meta tasks for tinning. The Boost library's arrays were used for output.

**Preliminary Testbed**

A testbed using Python and the windowing library WXWin, (WXWidgets) was attempted. It was observed that the WXWin bindings (WXWin is a C++ library) for Python are not very stable. A
side product was a GUI for the tinning program. The experimental testbed has a sluggish UI which is connected to a couple of C++ modules used to tin DEMs, render DEM files and render resulting TINs. A new testbed implementation is being built using Java, backed up by Matlab. This is justified by the fact that Java classes can be used natively within Matlab (Matlab’s own GUI is Java based) and Java provides a rich set of libraries for building GUI and for scientific tasks. The new testbed is scooping/segmentation oriented.

**Status**

We can process 10800x10800 arrays of posts in 1/2 hour on a PC. No external storage is used. Our test dataset was formed by catenating nine 3601x3601 cells of data from the Savannah March kickoff meeting.

The elevation range of the dataset is 3600. TINning it until the maximum absolute error was 100, i.e., 3%, produced 157,735 triangles in 1292 CPU seconds. TINning it until the maximum absolute error was 10, i.e., 0.3%, produced 5,320,089 triangles in 1638 CPU seconds.

**Lossless TIN**

TIN is inherently lossy, but we can make it lossless as follows.

1. Fit a surface to the TIN points, using any method.
2. Compute the errors.
3. Compress the errors, say with PAQ3N.

The more TIN points that are used, the more accurate the reconstructed lossy surface will be, and the less space will be needed to code the errors. The goal is to minimize the sum of the two sizes. Figure 4 shows the tradeoff when using the Laplacian PDE to interpolate the points. Other methods, such as ODETLAP, would be better.

![Tradeoffs with Lossless Compression with TIN](http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Geo...)
The \((x,y,z)\) TIN points must finally be coded. In this, note that the order of the set elements is immaterial. Bitmap coding techniques suffice for \((x,y)\). Assuming that there is not structure and that the points are random, there is an information theoretic limit. The coding of the \((x,y)\) now determines the order of the \(z\)s. They may be compressed with a sequence of Burrow Wheeler, run length and arithmetic coding.

**Scooping**

This representation is longterm research. The goal is to smash through the information theoretic barrier to terrain compression by utilizing geologic information. We are pursuing several representations in parallel.

Several different shaped scoops were tried by Inanc. The most primitive scoops are flat-bottom scoops (0-degree equations). They are very effective in representing areas like seas, oceans, lakes, plateaus and any naturally occurring flat areas. In contrast, a Morse description is totally at loss with those structures. Our initial efforts concentrated on working with square shaped drills. The problem with those is that natural terrain does not segment easily into squares. Thus to represent a lake many square shaped scoops are required. An improvement was developing hierarchical scooping akin quad-tree representation. Thus if a single square is too big it will result in a representation error beyond a threshold and thus will be subdivided internally into smaller squares. To decrease the number of scoops more elaborate models were tried. 1-degree scoops are still flat, however a full blown planes which can be tilted to represent slopes. 2-degree scoops use quadratic equations and are still more versatile. 3-degree scoops (cubics) use prohibitively many parameters (10 coefficients). However with the increase of the degree of scoops the number of scoop applications required to represent the terrain accurately decreases.

A next and natural step is to build a segmentation algorithm which can take care of the natural shapes of geological formations. A constructive algorithm for segmentation can start by growing segments around seeds of initially selected points. If a point coincides with a lake the segment can be expected to grow to the full lake size merging other segments thus representing the whole lake with a single 0-degree flat-bottom scoop. The problem here is representing shapes. There are different ways to represent 2D shapes. The simplest way is to use bitmaps. Those can be compressed using fax compression algorithms. More complicated methods use Voronoi diagrams and build a skeleton to represent shape.

Some of the tests will now be described in more detail.

**Degree-0 Scoops**

The purpose of this was to learn what types of drills are more useful by implementing a Matlab test program. It is a greedy optimization - at each step, select the drill instance and location

\[(\text{drill type}, x, y, z)\]

that removes the most material. Drill instances will generally overlap. If two drills drill the same post, then the minimum elevation is used. This nonlinearity is radically different from, and more
powerful than, wavelets. This nonlinearity also matches physical reality. If a river valley that cut 200' below the surrounding plane meets another river that is cut only 100' deep, then, at the confluence, the valley is only 200' deep, not 300'.

There are various subtleties of the greedy optimization. The drill that removes the most material at the start may be rendered completely redundant by later drills. This is reminiscent of stepwise multiple linear regression, where an initially important variable may ultimately be unimportant. A future possibility is to remove as well as add drills to the set. However this would be more compute intensive.

After the drills are determined, the final step would be encoding them. Note that the order of the drill applications is immaterial. Therefore, when coding, all the instances of the same drill may be grouped together. Then it is possible to sort the instances by location and delta encode the locations, since successive instances are relatively close. Alternatively a good bitmap compression method, such as CCITT group 4, could be used to code the drill locations.

The initial drills had no parameters other than their (x,y) location and (z) depth. One generalization, discussed in more detail later, is to allow more general (sloped) drills. This would tradeoff powerful, large to encode, basis elements, against small simple elements. However more of the simple more would be needed than of the complex ones. The sweet point would probably occur when the basis elements resemble the object being approximated. The purpose of these experiments is to better understand scooping, while initiating experiments in slope-preservation during lossy compression. The underlying assumption is that there is little long range correlation of elevation or slope.

An initial test on the 1201x1201 Lake Champlain West cell had these results. There were 11 drill types: a square of size 1 or 3, a circle of diameter 5, 10, or 20, a parabola of size 5, 10, or 20, and a flatter parabola of size 5, 10, or 20. After 10,000 drill instances, the initial material volume of $1.9 \times 10^9$ was reduced to $4.2 \times 10^7$. The mean absolute elevation among the 1201$^2$ posts was 30m, or 2% of the elevation range. 98% of the selected drill instances were the diameter 20 circle, with the diameter 10 circle and size 3 square distantly behind. The parabola drills were not selected even once. Table 1 gives the complete results.

<table>
<thead>
<tr>
<th>Drill Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square, D=1</td>
<td>0</td>
</tr>
<tr>
<td>Square, D=3</td>
<td>35</td>
</tr>
<tr>
<td>Circle, D=5</td>
<td>2</td>
</tr>
<tr>
<td>Circle, D=10</td>
<td>129</td>
</tr>
<tr>
<td>Circle, D=20</td>
<td>9834</td>
</tr>
<tr>
<td>Parabola, size 5</td>
<td>0</td>
</tr>
<tr>
<td>Parabola, size 10</td>
<td>0</td>
</tr>
<tr>
<td>Parabola, size 20</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 5</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 10</td>
<td>0</td>
</tr>
<tr>
<td>Flatter parabola, size 20</td>
<td>0</td>
</tr>
</tbody>
</table>

http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Geo... 02/16/09 20:49
We do not propose this specific set of drills as a polished technique to use, yet, but as an example of the experiments that we are conducting in order better to understand this idea.

**Degree-1 Scoop**

This version differs in the following ways. Instead of a set of overlapping drills, selected in a greedy manner, here there is only one drill, a $7 \times 7$ square, and the footprints of its applications partition the dataset. The face of each scoop is a tilted plane whose slope is chosen to minimize the elevation error.

Therefore, a $7 \times 7$ Scoop size represents 49 elevations using only 3 coefficients. 7 is not a magic number, and indeed we have experimented with various sizes. However, 7 is good enough for Level 2 DTED cells: Large errors are rare and the mean error is very low, often less than 2 meters. The regularity brings simplicity to this representation.

Here are some test results on the W111N31 DTED II cell, shown in Figure 5, with about $3600^2$ posts with elevations from 811 to 2882, shown below. (To simplify the programming in these experiments, a few rows and columns were shaved off the sides of the cell to make the size an exact multiple of the tile size.)

![W111N31 Cell](image)

Figure 6 shows the effect of using fewer and larger tiles to approximate the cell. Even with $7 \times 7$ tiles, about 90% of the posts are represented to an absolute error under 4, or 0.2% of the elevation range.
Percent of Posts with Errors Under Certain Cutoffs When Cell Approximated with Tiles of Various Sizes

Another test was performed with 5x5 drills, optimized with Singular Value Decomposition (SVD), on the Lake Champlain West cell, with a data range of 1576. The maximum error was 101, but that was rare. Over 50% of elevation posts were exactly represented. (This was helped by Lake Champlain being large and flat.) The input data size was 1442401 numbers, and the output data size 172800 numbers, for a compression ratio of 8.35:1. The contour images in Figure 7 contrast the original and reconstructed terrain. We see no visible difference.

Lake Champlain W Cell: Original and Represented by 5x5 Drills (Data Reduction Factor 8.35)

For another example, we used 7x7 drills to represent a 100x100 detail of the W111N31 cell, with an elevation range of 512. The reduction in data size is a factor of 16.3:1. Figure 8 shows the error histogram.

Next, we used observer viewsheds to evaluate this representation, as follows.

1. Pick 81 observers,
2. Assume that each observer is elevated 10 units above the terrain elevation at its location.

3. Assume that each observer can see out to a radius of interest that is 300 posts.

4. Assume that potential targets are also 10 units above their local terrain.

5. For each observer, compute its viewshed, or what posts it can see, on the original dataset.

6. Redo the computations on terrain reconstructed with the alternate representation.

7. Measure the area of the symmetric difference between the two sets of viewsheds, as a fraction of the cell's area.

<table>
<thead>
<tr>
<th>Representation % Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>tile3</td>
</tr>
<tr>
<td>tile5</td>
</tr>
<tr>
<td>tile7</td>
</tr>
<tr>
<td>tile75</td>
</tr>
</tbody>
</table>

The tile3 representation approximates the surface with a regular grid of 3x3 tiles. tile5 and tile7 are similar. The tile75 representation uses first 7x7 and the 5x5 tiles.

Figure 8 and Figure 9 compare the joint viewsheds for the original terrain and the tile7 representation. The visible difference is small.

**Sculpting**

This is an extension of scooping being studied by Westort. She adapted a generic DTM sculpting tool definition and implementation from Java and C++ to Matlab, including blade, path, and target model matrices. With the help of Tracy and Luk, she significantly improved the efficiency of the code for large datasets. She evaluated the use of Arc/Map drainage and slope functionality for adoption as paths and blades, respectively. Together with Steve Martin, she is developing a stand-alone hydrological modeling solution for path and blade generation for DTM compression and reconstruction.
Error Histogram from Tiling 100x100 Detail of W111N31 with 7x7 Drills

Joint Viewsheds on the Original Terrain
Joint Viewsheds on the Tile7 Representation

This subproject has the largest potential payoff, for several reasons. It will be able to represent discontinuities (cliffs), and its nonlinearity is powerful. Dr Westort is creating an initial testbed in Matlab to help us understand the idea. This initial testbed will be of the nature of a terrain reconstruction assistant. The user will specify the terrain, the X-Y paths that the scoop (also called a blade) will follow, and the blade's circular cross-section. The system will run the blade along the path at the lowest elevation that does not cut into the terrain. It will show the error at each post and report the average and max error. We anticipate demonstrating a prototype at the review meeting in San Diego.

This testbed will provide a facility to gain experience with different blade shapes, including variable shapes. That experience will allow us to freeze some decisions, such as whether to use variable shapes, or not. The former choice will require fewer cutting paths, but with the latter choice, each path will take less space to represent. An analogy to splines would be, what is the appropriate polynomial degree to use?

Cutting paths are appropriate for canyon-like terrain. However, reconstructing, say, a mesa, might be better done with other operators. Designing such operators will be the next step, after we can reconstruct eroded terrain.

Postprocessing the reconstructed terrain with a smoothing operator might improve the accuracy at the cost of adding only one bit to the size of the representation ("apply the builtin smoother"). Smoothing will effective if the terrain satisfies some smoothness criterion, which contradicts our design principle that terrain is often discontinuous. Nevertheless, for completeness, this should be checked.

Finally, after obtaining a sufficient understanding of this idea, we might think about formulating and proving some theorems about it.

ODETLAP (Overdetermined Laplacian PDE)

This surface representation and reconstruction technique was pioneered by Franklin, and is

http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Geo...
being investigated in this project primarily by Xie. It extends the traditional Laplacian (heat flow) partial differential equation (PDE), which interpolates unknown points in a matrix of known and unknown elevation posts, as follows. A system of linear equations is constructed with one equation for each unknown elevation, except those in the first or last row or column of the matrix:

\[ 4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} \]

Choosing appropriate equations for the unknown border points is tricky, for some uninteresting technical reasons. The resulting sparse system easily can be solved by, e.g., Matlab, for 1201x1201 matrices. We call this technique DETLAP for exactly DETermined LAPlacian system. ODETLAP extends DETLAP as follows.

1. One instance of the above equation is created for every (nonborder) point, whether known or unknown.
2. Each known point also induces a second equation:
   \[ z_{ij} = h_{ij} \]
3. Each Border point induces a slightly different equation or two.

The resulting system of linear equations is overdetermined, i.e., inconsistent, so no exact solution is generally possible, but only a least square approximation, as follows. If the system is

\[ Ax = b \]

we instead solve

\[ Ax = b + e \]

so as to minimize the error \( e \). The solution is

\[ x = (A^tA)^{-1}A^tb \]

The two types of equations may be given different relative weights, depending on the relative importance of accuracy versus smoothness. Matlab easily processes 400x400 matrices of elevation posts. More specialized techniques can handle larger systems. See [^John Childs masters, A Two-Level Iterative Computational Method for Solution of the Franklin Approximation Algorithm for the Interpolation of Large Contour Line Data Sets, http://www.terrainmap.com/ ^], which uses a Paige-Saunders conjugate gradient solver with a Laplacian central difference approximation solver for the initial estimate.

ODETLAP has many advantages, which are not, to our knowledge, combined in any other surface approximation technique.

1. It will infer a local maximum, i.e., a mountain top, inside a set of nested contours.
Therefore, it does not generate artificial buttes all the time.

This property can be surprising because points interpolated with a Laplacian PDE must fall inside the range of boundary value points. However, we're not doing an exact Laplacian PDE, but an extension of it. Therefore, we can generate values outside the range of boundary value points.

2. The generated surface doesn't droop between contours. For reasonable weighting parameters, the contours are invisible in the resulting surface.

3. It will utilize isolated data points, if available.

4. It will interpolate broken contours. This is not a property of methods that fire out lines until they hit a contour.

5. The resulting surface is at least approximately conformal. That is, if a conformal mapping is applied to the input points the resulting surface is that conformal map of the surface computed from the untransformed points. Therefore, sets of nested kidney-bean contours are handled reasonably. Again, this property is not shared by all surface approximation methods.

6. ODETLAP will conflate inconsistent data, with user-defined weights, producing one merged surface. This allows one to combine a large, low-resolution dataset with one or more small high-resolution datasets covering only parts of the large dataset.

Figure 11 shows a stress test of ODETLAP. The test data is a set of four nested square contours. A surface is fit to them with ODETLAP run three times with different values of the smoothness parameter, $R$. Larger values for $R$ mean a smoother but less accurate surface. The surfaces are plotted with an oblique projection so that the silhouette edge is obvious. The lower right surface shows no evidence at all of the original square contours, and interpolates a smooth mountain top in the center. The maximum absolute elevation error is 12% of the elevation range, and the mean only 2.7%. These errors would be much smaller for smoother contours.
Nested Square Contours Generating Three ODETLAP Surfaces with Varying Smoothness and Accuracy

The impact of ODETLAP for this project is that surprisingly few points are needed to generate a good surface.

We tested ODETLAP on a mountainous 400x400 section of the Lake Champlain W DEM, with elevation range 1378m, by selecting every $K$th post in both directions and using ODETLAP to fit a surface (with $R=0.1$). The following table shows that average absolute errors. Data set size $%$ is the ratio of posts used to reconstruct the ODETLAP surface compared to the original number. It is $1/K^2$.

<table>
<thead>
<tr>
<th>$K$</th>
<th>Data set size, $%$</th>
<th>Error</th>
<th>Error $%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>11.1%</td>
<td>0.9</td>
<td>0.065%</td>
</tr>
<tr>
<td>4</td>
<td>6.25%</td>
<td>1.6</td>
<td>0.11%</td>
</tr>
<tr>
<td>5</td>
<td>4%</td>
<td>2.4</td>
<td>0.17%</td>
</tr>
<tr>
<td>7</td>
<td>2%</td>
<td>4.1</td>
<td>0.29%</td>
</tr>
<tr>
<td>10</td>
<td>1%</td>
<td>6.5</td>
<td>0.46%</td>
</tr>
<tr>
<td>15</td>
<td>0.44%</td>
<td>10</td>
<td>0.71%</td>
</tr>
<tr>
<td>20</td>
<td>0.25%</td>
<td>12</td>
<td>0.86%</td>
</tr>
<tr>
<td>30</td>
<td>0.11%</td>
<td>14</td>
<td>1.0%</td>
</tr>
<tr>
<td>50</td>
<td>0.05%</td>
<td>16</td>
<td>1.1%</td>
</tr>
</tbody>
</table>

That is, with ODETLAP we reduced the data size by a factor of 100, at a cost of an average error of 0.46%.

**Choices for Input Points**

Xie has have been mainly working on ODETLAP in the past year. To be more specific, he is
trying different sampling methods on some 400x400 elevation matrices and then using ODETLAP to reconstruct the surface.

We have tried three different sampling methods.

1. The first technique, the regular method is just regular sampling, i.e., choosing 1 point for each K points in both horizontal and vertical directions. The compression ratio is $1/K^2$.

2. The second technique, the TIN method, is to use output points from the TIN program, which iteratively selects points furthest from the current surface.

3. The third technique, the improved TIN method, operates as follows.

   a. Run TIN to produce a predetermined number of important points.

   b. Use ODETLAP to reconstruct the surface from those points. Note that the surface will pass near, but not exactly through, the points.

   c. Find the worst points in that surface, that is the points whose reconstruction is farthest from their correct value.

   d. Add those points to the set of points from TIN, and rerun ODETLAP to compute a better surface.

Here is a comparison of the three methods on three 400x400 test cells, with ODETLAP smoothness parameter R=0.3. Both the regular and TIN methods used 1000 points. The refined TIN method added the 100 worst points from the TIN method. For each test, we measured the average absolute error and the maximum absolute error. The column headings are as follows.

<table>
<thead>
<tr>
<th>Abbrev</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIN a</td>
<td>TIN, average error</td>
</tr>
<tr>
<td>Reg a</td>
<td>Regular, average error</td>
</tr>
<tr>
<td>Ref a</td>
<td>Refined TIN, average error</td>
</tr>
<tr>
<td>TIN m</td>
<td>TIN, max error</td>
</tr>
<tr>
<td>Reg m</td>
<td>Regular, max error</td>
</tr>
<tr>
<td>Ref m</td>
<td>Refined TIN, max error</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>Reg a</th>
<th>TIN a</th>
<th>Ref a</th>
<th>Reg m</th>
<th>TIN m</th>
<th>Ref m</th>
</tr>
</thead>
<tbody>
<tr>
<td>W111N3110</td>
<td>3.4</td>
<td>5.6</td>
<td>5.3</td>
<td>98</td>
<td>31</td>
<td>27</td>
</tr>
<tr>
<td>W111N3111</td>
<td>9.0</td>
<td>10.6</td>
<td>10.4</td>
<td>134</td>
<td>113</td>
<td>66</td>
</tr>
<tr>
<td>W111N3112</td>
<td>1.7</td>
<td>2.7</td>
<td>2.6</td>
<td>114</td>
<td>20</td>
<td>16</td>
</tr>
</tbody>
</table>

The TIN method produced a somewhat worse average error, but a much better maximum error. That is, the TIN method produced a much better conditioned surface. The refined method was
slightly better than the TIN method under both the average and maximum metrics. Modifying the refined method to add even more bad points would presumably make it even better. That is an area for future research.

Figures 12 to 14 compare the regular and TIN methods, each with 100 points, on the W111N3110 dataset. Note how the TIN method captures small features better. Note also how well even 100 points represent the original surface.
Figures 15 to 17 show an example of approximating a more complex surface with fewer points. Even only 36 TIN points capture the broad essence of the surface.

Original W111N3111 Dataset (160,000 Points)
We have also done some preliminary experiments on a new sampling method, namely, the visibility method. This method fits a surface to the points of greatest visibility.

**Filling in Missing Data Holes**

This topic was pursued because of sponsor interest expressed at the review meetings.

Terrain (elevation) data sometimes has holes or regions of missing data, perhaps because of the production technology. This section compares some algorithms for filling in a circle of radius r=100 of missing data. This missing region is large enough to stress-test the various
algorithms, most of which are unacceptably bad. Any algorithm that succeeds here can also handle smaller holes.

The two best algorithms, which produce realistic results, are a thin plate PDE, and an overdetermined Laplacian PDE. The thin plate PDE is faster, while the overdetermined Laplacian PDE may also be used to smooth known data.

The test data and environment are as follows.

1. The terrain format is an array of elevations.

2. The test data sets are drawn from the 1201x1201 Lake Champlain West USGS level 1 DEM, seen in Figure 18. It combines part of the Adirondack Mountains, the highest in NYS, with Lake Champlain and its lowlands. North is to the right.

   Lake Champlain West DEM

1. The region of missing data, which needs to be interpolated, is always a circle of radius r=100.

2. Each test missing circle was embedded in a 400x400 square of data. That is relevant only for the two interpolation algorithms that use data beyond the boundary of the missing data. Those are the two best algorithms, overdetermined Laplacian PDE and thin plate PDE. They actually use data within 2 posts of the circle of missing data. The poorer algorithms use only data on the circle.
Figure 19 nicely summarizes the results. The data outside the black circle (r=100) is real; the data inside was interpolated by our overdetermined Laplacian PDE. Although the thin plate PDE produces similar results, our method has other advantages that that lacks, such as the ability to conflate inconsistent data.

Using ODETLAP to Fill in the 31416 Missing Posts Inside the Circle

1. Note the realistic appearance of the contour lines.

2. Note the interpolated local mountain top near the bottom just inside the circle. That was caused by the large slope of the known terrain just outside the circle. This interpolated feature is remarkable since few interpolation algorithms can produce a value that is outside the range of the known values.

More detailed results are available at a web site with a javascript program allowing the user to choose from various preprocessed datasets, [http://www.ecse.rpi.edu/Homepages/wrf/Research/geostar-reports/fill_data_holes/index.html]. That presents the results of 6 interpolation algorithms on 20 different sections of Lake Champlain West. Each cell of the table shows six algorithms on one dataset:

R=.1 overdetsol()
This is our overdetermined Laplacian PDE. R=.1 means that the equations for all points being equal to the average of their neighbors are weighted 0.1 as much as the equations for the known points being equal to their known values. That results in no visible smoothing of the known data.

detsol()
This is the Laplacian PDE for the unknown points.
thinplate()  
This is the thin plate PDE for the unknown points.

Matlab nearest  
This is the Matlab builtin nearest point interpolation algorithm, chosen because it is well known.

Matlab linear  
This is the Matlab builtin linear interpolation algorithm.

Matlab cubic  
This is the Matlab builtin cubic interpolation algorithm.

A fourth Matlab interpolation algorithm is not shown since it failed. Figure 20 compares the different techniques.

Comparing Six Methods to Fill in a Missing Data Hole

Planned future work on this topic is as follows.

1. Using the builtin Matlab overdetermined linear equation solver for our overdetermined Laplacian algorithm is very slow. We are currently researching faster solution techniques.

2. This algorithm can interpolate general classes of data, such as contour lines, possibly broken, and isolated points, perhaps derived from a TIN. We plan to study this in more detail.

Multiobserver Siting
This is prototype software originally written by Franklin, and extended Christian Vogt as a masters thesis. It is a set of C++ programs that exchange data in files and are controlled by Linux shell scripts and makefiles. The toolkit works, albeit being somewhat complicated and messy. It can quickly compute high resolution viewsheds as demonstrated in Figure 21.

![Sample Viewsheds](image)

Dan Tracy is using the siting toolkit to evaluate our compression. The following figures demonstrate the toolkit's output. The test dataset is the Lake Champlain West DEM. A particular radius of interest and observer and target height are picked and the 60 observers are sited, so that their viewsheds will jointly maximally cover the cell. See Figure 22. Figure 23 adds an *intervisibility* requirement. That means that the observers must form a connected graph, if there is a graph edge between pairs of observers that can see each other. This allows messages to be passed from observer to observer, which is useful if they are radio base stations.
Siting 60 Observers on Lake Champlain W, WITHOUT Intervisibility

Siting 60 Observers on Lake Champlain W, WITH Intervisibility

http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Geo...
Of course, the area of the joint viewshed is smaller since the individual viewsheds must overlap, as quantified in Figure 24.

![Graph showing joint viewshed comparison](image)

**Effect on Joint Viewshed of Requiring Intervisibility as the Number of Observers is Increased**

RPI has issued a subcontract to Environmental Systems Research Institute, Inc. to productize our siting toolkit it as an ArcGIS module. A copy of the contract was sent to DARPA/DSO to check that the government will be able to use this module for free.

**Effect of Reducing Resolution**

We earlier conducted initial studies of the effect of lowering the horizontal or vertical resolution of the dataset before performing the multiobserver siting. We observed that lowering the horizontal resolution lowers observer siting quality. However, lowering the vertical resolution does not have as large as effect.

The visibility, computed on the lower resolution data, is too high. In other words, when an observer's viewshed is computed on a lower resolution matrix of elevation posts, it may have a larger area (after scaling) than when computed on the original, high resolution, data. That means that it is important to compute the viewsheds on the best available data.

Careful consideration was given to the proper procedure for evaluating the effect of reduced resolution; the chosen procedure is given in Figure 25.
Our point is this. When evaluating a set of observers, it does not matter where they are, but how much they can see. There may be many sets of observers, each as good as the other. Therefore, when performing a multiple observer siting on the reduced resolution dataset, we compute their joint viewshed, and compare its area to that of the joint viewshed of multiple observers sited on the original, best quality, data. Also, when computing the viewshed area of the observers sited on the lo-res data, we do the computation on the hi-res data, since that is more accurate.

Figures 26 and 27 show some experiments with reducing the horizontal and vertical resolution. The visibility index is the fraction of the cell covered by the joint viewshed of a fixed number of observers sited with our testbed. The "hi res" line shows the result of using the original data. The lo-res line shows the siting and viewshed computation formed on the lo-res data, for three different reductions in horizontal resolution. The transferred res line shows the effect of siting the observers on the lo-res data, but then computing their joint viewshed on the hi-res data. It is this line that should be compared to the hi res line. The extent to which it is smaller shows the deleterious effect of reducing the data resolution.

The difference between the lo res and transferred res lines shows the error caused by evaluating viewsheds on lo-res data. That the lo res line is higher shows that the error is biassed up. There may be no deep significance to this; it may be an artifact of how we interpolate elevations between adjacent posts.
Figure 28 shows how imprecise viewshed computations can be. The test data is the Lake Champlain West cell; the observer is near the lower left corner, on Mt Marcy. A particular observer and target height are chosen, and then the visibility of each point in the cell is computed by running lines of sight from the observer. When the line runs between two posts with elevations $A$ and $B$, and some elevation must be computed there, one of four rules is used.

1. $\min(A,B)$
2. $\text{mean}(A,B)$
3. linear interpolation between $A$ and $B$ depending on their relative distances from the line of sight.
4. $\max(A,B)$
The choice of interpolator affects the visibility of "one half" of all the points in the cell. The black region is hidden for all the interpolators. The dark grey region is hidden for three interpolators, the light grey region for one, and the saturated color region is visible for all of the interpolators. In other words, the visibility of (at least!) one half of the cell is actually unknown.

Smugglers' Path Planning

A path planning program was created by Tracy. Given a cost metric for the terrain, a start point, and an end point, the program computes the path between the two points that minimizes the line integral over the cost metric by implementing the A* algorithm. In particular, with a strictly binary go/no go cost metric (e.g. avoid the area visible by the planted observers,) the Euclidean distance of the path is minimized. We call this smugglers' path planning.

Future work may entail computing other error metrics related to the visibility and path planning.

Using Smugglers' Path Planning to Evaluate Alternate Representations

The purpose for researching alternate terrain representations is to use the terrain for some application, such as path planning. Since the alternate representation is good only so far as it supports such applications, we have started to use them to evaluate our representations, in addition to using elevation error.

The assumption is that the cost of a path between some source and goal depends on the how visible that path is to a set of observers. If our client is a smuggler, then low visibility is
desirable. Conversely, if our client is using a radio and the observers are base stations, then high visibility is good.

Initially, we are computing paths that stay out of the joint viewshed of a set of observers optimally sited to see as much as possible of the terrain. Finding no suitable implementation on the web, we implemented this ourselves. (The main difficulty is that the cost matrix is large and sparse. A second problem is that it is impossible explicitly to compute and store the distance between every pair of mutually visible points; remember that we are not working with toy-sized datasets.) Here is one test of our techniques.

Figure 29 shows the test dataset, W111N31, 3595x2595 posts with 12,924,025 coefficients (elevations). The elevation range is 2071.

The alternate representation is to use 7x7 scoops. The reduced the number of coefficients to 791,267, that is, by a factor of 16:1. The mean absolute elevation error was 1.7, or 0.1%. There is no visible difference in the image of this representation.

Next we optimally sited 324 observers, with radius of interest 100, to maximize their joint viewshed, shown in Figure 30.
Joint Viewshed of Optimally Sited Observers on the Tile7 Terrain Representation

Then we picked a source and goal that would have a complicated path between them, computed the shortest path that avoided the joint viewshed, and plotted the terrain, viewsheds, and path together in Figure 31.
Our first evaluation of the alternate representation went as follows. We transferred the 324 observers back to the original terrain and recomputed their viewsheds. Then we overlaid the path on them and counted how many path pixels were inside the more accurate viewsheds. The path was 4767 points long; only 14, or 0.3% were visible, as shown in the Figure 32.
Coding the Representation's Coefficients

The representations discussed in this report all reduce the terrain to a set of coefficients. The raw dataset is a matrix of elevations. The degree-1 scoops are triples of coefficients of planes, and so on. As a final step, these coefficients must be coded, or reduced to a minimal number of bits. There is a wide range of coding research to draw on, such as JPEG when the coefficient matrix looks like an image. However, we hope to do better. This is future research.

Comparison of Actual Achievements with Goals

This project's progress is aligned with the proposal except for the following change. Because of sponsor feedback, we are researching path planning, instead of hydrology.

Cost Overruns

None.
Interactions Between RPI and Sponsors

In addition to written reports, there have been several face-to-face meetings during this contract.

1. March 2005: kickoff meeting in Savannah GA, including presentation by WR Franklin.
2. June 2005: site visit by Paul Salamonowicz and Ed Bosch of NGA to RPI.
3. Nov 2005: Program review meeting in San Diego CA, including presentation by WR Franklin.
4. April 2006: Program review meeting in Snoqualmie WA, including presentation by WR Franklin.

In addition, a password-protected website has been established, [http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/GeoStar/], to communicate results in a less formal manner.

Publications

Papers presenting results from this project have already been accepted at the following scientific meetings.


More papers are in preparation for other meetings and then journals.

Press and Blog Mentions

With the prior approval of DARPA, RPI announced the contract with a press release that was mentioned in the following RPI places.
1. RPI Press release, 10/31/2005
2. RPI School of Engineering Press Release
3. Rensselaer School of Engineering News
4. Improving Terrain Maps, Rensselaer Alumni Magazine Winter 2005-06

That press release was picked up by these news sources.

1. Better terrain maps of Earth... and beyond, in Roland Piquepaille's Technology Trends, 5 nov 2005
2. ZDNet, 11/5/2005
3. Surf wax Government News
4. Defence Talk, 11/1/2005
5. Advanced Imaging (no longer online)
6. ACM Technews 7(862), 11/2/2005
7. Interview on WGY AM-810 radio, 11/3/2005

Technical Notes

This report was written as a pmwiki wiki page, on http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/GeoStar/June2006AnnualReport. Contact Franklin for the password. The wiki is the best place from which to copy the text. The images are stored at http://www.ecse.rpi.edu/Homepages/wrf/wiki/GeoStar/many in both lo-res and hi-res versions. Then it was converted to a PDF file with http://www.wikipublisher.org/.

[*#*]

Page last modified on October 01, 2006, at 10:03 PM
Compact Visibility and Path Preserving Terrain Representations

Prof. W Randolph Franklin, Prof. Caroline Westort, Prof. Frank Luk, Metin Inanc, Zhongyi Xie, Dan Tracy, and Steve Martin

RPI

ECSE Dept, 6003 JEC, 110 8th St, Troy NY 12180
mailATwfranklin.org, +1 518 276 6077; http://wfranklin.org

@RISK: The Consensus Security Vulnerability Alert
Vol. 5 No. 33 (8/21/2006)

Affected: Microsoft PowerPoint, possibly all versions

Description: A remote code execution vulnerability, which could either be different from the one patched by MS06-048 or a variation of it, has been reported in Microsoft PowerPoint. The flaw is being exploited by some Trojans in the wild. The technical details about the vulnerability have not been publicly posted yet.

Status: Microsoft has not confirmed, no updates available. Users should be advised to refrain from opening PowerPoint from unknown sources.
Goal

Lossily compress terrain datasets

Evaluation:
- Multiple observer siting
- Path planning to avoid the observers’ viewsheds (smuggler’s path planning)

Old Results Review

Incremental TIN on 10800x10800 data: produces K most important points

Fill in missing data holes of size 200

Today: more focused; improvements; Coding of coefficients.
Test Data
(400x400 DTED II)

W111 N31 subsets
  Hill1
  Hill2
  Hill3

W121 N38 subsets
  Mtn1
  Mtn2
  Mtn3

Test Data Stats

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Elev Range, meters</th>
<th>Gzipped size, KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1</td>
<td>505.</td>
<td>79.</td>
</tr>
<tr>
<td>Hill2</td>
<td>745.</td>
<td>134.</td>
</tr>
<tr>
<td>Hill3</td>
<td>500.</td>
<td>59.</td>
</tr>
<tr>
<td>Mtn1</td>
<td>1040.</td>
<td>177.</td>
</tr>
<tr>
<td>Mtn2</td>
<td>953.</td>
<td>174.</td>
</tr>
<tr>
<td>Mtn3</td>
<td>788.</td>
<td>174.</td>
</tr>
</tbody>
</table>

Original binary size = 320KB each
Alternate Representations

- Drainage scooping
- Region-based Scooping
- Regular scooping
- ODETLAP

Terrain is Nonlinear

- In the real world, features do not linearly superimpose onto each other. *(Two canyons can’t cross.)*
- So, we strive for nonlinear representations to match the geology.
Why an Underlying Regular Grid?

- Starting with a regular data grid saves work.
- And compats with the existing data.
- Grids are easy to refine.
- Our representations do compress adaptively according to the local complexity.
- Using a higher resolution does not greatly increase the size if there is no new info (e.g., features).

Alternate Representations

- Drainage scooping
- Region-based Scooping
- Regular scooping
- ODETLAP

Longterm

Shortterm
Drainage Scooping Overview

- Breaks complex terrain into more simplistic parts
- Utilizes hydrological characteristics to
  - Discover those parts
  - Assist in encoding
- Different compression methods for different parts

Strategy

Terrain map

Build stream networks

Replace some catchments with planes

Replacement log

Intermediate height map

Yes

Should more replacements be made?

No

Simplified height map
Stream networks

Links made to neighbor of minimum relative elevation

Height
Link direction
Node

Data writing order for a replacement log record

Recursion levels

Replacement Criteria

Stream networks are replaced with planes such that there is a:
- Reduction in the number of stream networks
- Increase in the planarity of stream networks
**Status**

- **Complete:** Stream network discovery
- **Complete:** Stream network replacement
- **Partially complete:** Stream network replacement criteria and encoding

**Alternate Representations**

- Drainage scooping
- **Region-based Scooping**
- Regular scooping
- ODETLAP

**Longterm**

**Shortterm**
Region-Based Scooping

- Partition plane into irregular quasi-planar regions
- Start: 40000 2x2 seeds
- Each region grows as feasible
- (Now they overlap a lot.)
- Pick biggest region
- Loop: include region with most area not already included.
- Status: In development

Alternate Representations

- Drainage scooping
- Region-based Scooping
- Regular scooping
- ODETLAP

Longterm

↓

Shortterm
Regular Scooping Algorithm

Original terrain rep → Partition into 7x7 planar scoops → (z=ax+by+c) → Nonlinearly transform a,b → Bzip2 a,b, Jpeg c → Scooped representation

Reg Scoop Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Orig Size, KB</th>
<th>Gzip size, KB</th>
<th>Scoop size, KB</th>
<th>Gzip / scoop size</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>320</td>
<td>79</td>
<td>14.5</td>
<td>5.45</td>
<td>1.75</td>
</tr>
<tr>
<td>hill2</td>
<td>320</td>
<td>134</td>
<td>15.6</td>
<td>8.58</td>
<td>3.51</td>
</tr>
<tr>
<td>hill3</td>
<td>320</td>
<td>59</td>
<td>12.7</td>
<td>4.66</td>
<td>0.99</td>
</tr>
<tr>
<td>mtn1</td>
<td>320</td>
<td>177</td>
<td>15.7</td>
<td>11.3</td>
<td>7.02</td>
</tr>
<tr>
<td>mtn2</td>
<td>320</td>
<td>174</td>
<td>15.6</td>
<td>11.1</td>
<td>7.07</td>
</tr>
<tr>
<td>mtn3</td>
<td>320</td>
<td>174</td>
<td>15.7</td>
<td>11.1</td>
<td>7.04</td>
</tr>
</tbody>
</table>
Regular Scooping Results

Visibility and Path Evaluation

- Observer/target heights = 10 m
- Radius of interest = 30
- Number of observers = 64
- Path cost metric:
  - Traversing 1 hidden post = 1
  - Traversing 1 visible post = 100
  - Both multiplied by a strong uphill cost factor.
Path Planning Algorithm

- Planning paths around obstacles is harder than on a smooth surface.
- It’s unclear how smoothly to optimize a geodesic because of local optima.
- We use an A* search.
- Path planning is relatively easy.

Path Planning Images

- In following slides:
- Right image is path planned on the alternate scooped representation
- Left image recomputes viewsheds and replans path, using right’s observers.
- The purpose is to check the alternate representation’s path’s optimality.
### Scooping Stats

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Cum Viewsed Error</th>
<th>Path Length Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1</td>
<td>0.54%</td>
<td>1.14%</td>
</tr>
<tr>
<td>Hill2</td>
<td>1.49%</td>
<td>0.35%</td>
</tr>
<tr>
<td>Hill3</td>
<td>0.05%</td>
<td>0.46%</td>
</tr>
<tr>
<td>Mtn1</td>
<td>6.36%</td>
<td>12.1%</td>
</tr>
<tr>
<td>Mtn2</td>
<td>6.94%</td>
<td>11.3%</td>
</tr>
<tr>
<td>Mtn3</td>
<td>6.99%</td>
<td>17.6%</td>
</tr>
</tbody>
</table>

### Scooping Conclusions

- All numbers here are means of the 6 observations.
- Compression ratio: 21.3 relative to the original binary format, or 8.7 relative to gzip.
- RMSE: 4.6 m.
- Cumulative viewshed error: 3.7%
- Optimal path length error: 7.2%
Regular Scooping
Future

- More sophisticated scooping
- Hybridize different scoops
- Consider dynamic scoops & methods
- Consider scattered points

Alternate Representations

- Drainage scooping  
  Longterm  
- Region-based Scooping  
- Regular scooping  
- ODETLAP  
  Shortterm
ODETLAP Properties

- Extension of Laplacian PDE
  - Every nonborder point the average of its neighbors
  - Known points equal to their known values
- Sparse overdetermined linear system
- \( \{x,y,z\} \rightarrow z(x,y) \)
- Infers local maxima ✓
- Conflates inconsistent data ✓
- Input points are invisible in surface ✓
- Matlab version uses memory and cycles
- Hence the 400x400 datasets for now.

ODETLAP Algorithm

Original terrain rep → Incremental TIN → Set of important points

ODETLAP → Reconstructed surface

Some worst (and sparse) points → Augmented important point set

Refine? → n

y
ODETLAP Refinement

![Graph showing refinement process]

Observation: Points are Clustered

![Map showing clustered points]

ODETLAP Improvement

- When adding a new bad point to the important point set...
- Don’t add a point near an earlier bad point added in this step.

ODETLAP Coding Method

To represent \{(x, y, z)\} compactly

- **Observe:** \{(x, y)\} form bitmap.
- Bzip2 the PGX file.
- **Note:** this beats JJ2000.
- Z doesn’t compress – just strip hi 0s
Estimated ODETLAP Coding Sizes

<table>
<thead>
<tr>
<th>N Pts in Representation</th>
<th>Original Size</th>
<th>N Bytes to Encode (x,y)</th>
<th>N Bytes to Encode z</th>
<th>Total ODETLAP Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>320,000</td>
<td>234</td>
<td>111</td>
<td>345</td>
</tr>
<tr>
<td>300</td>
<td>320,000</td>
<td>544</td>
<td>333</td>
<td>877</td>
</tr>
<tr>
<td>1,000</td>
<td>320,000</td>
<td>1,284</td>
<td>1,111</td>
<td>2,395</td>
</tr>
<tr>
<td>3,000</td>
<td>320,000</td>
<td>2,893</td>
<td>3,333</td>
<td>6,226</td>
</tr>
</tbody>
</table>

Visibility and Path Evaluation

- Observer/target heights = 10 m
- Radius of interest = 30
- Number of observers = 64
- Path cost metric:
  - Traversing 1 hidden post = 1
  - Traversing 1 visible post = 100
  - Both multiplied by a strong uphill cost factor.
Evaluation

- When we site the multiple observers on our alternate representation, how accurate is the computed area of the observers’ cumulative viewshed?
- When we plan a path avoiding the viewsheds, is this path indeed hidden?
- We selected parameters to make the optimal paths as complicated as possible.
ODETLAP Viewshed and Path Expts.

<table>
<thead>
<tr>
<th>N Pts in repr</th>
<th>Size, bytes</th>
<th>Compression ratio rel to gzip</th>
<th>Mean % viewshed error</th>
<th>% Path visible (lower better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>345</td>
<td>386</td>
<td>15%</td>
<td>0.34%</td>
</tr>
<tr>
<td>300</td>
<td>877</td>
<td>152</td>
<td>13%</td>
<td>0.78%</td>
</tr>
<tr>
<td>1000</td>
<td>2395</td>
<td>56</td>
<td>7%</td>
<td>0.77%</td>
</tr>
<tr>
<td>3000</td>
<td>6226</td>
<td>21</td>
<td>3.25%</td>
<td>0.44%</td>
</tr>
</tbody>
</table>

Comparison to JPEG

- **Note:** Which JPEG? JPEG, lossless JPEG, JPEG-LS, JPEG 2000 (jasper or jj2000), …
- Our results are already comparable to JPEG2000 representations of the same size, but
- We’re still improving.
Deliverables

• SW to represent terrain using
  – ODETLAP
  – Scooping
• ArcGIS DLLs for multiobserver siting.

Future

• Scooping and ODETLAP are wide open topics.
• Many many unexplored avenues.
• Global system opts
• Apply ODETLAP to highly visible points,
• Lossily compress ODETLAP (x,y).
• Many scooping possibilities
Smugglers’ Path Planning on 16x Compressed “Scooped” Terrain Representation

Original: 3955x3955
WinNN1 Terrain
12,824,265 d.f. Elev.
Range=2071

Compressed (1/7 Scoop): 1,791,297
d.f. (15x reduction). Mean abs.
error=1.7 (0.1%).

Compressed: Shortest
Smugglers Path Computed
Avoiding All 324 Viewsheds of
Optimally Sited Observers

Original: Viewshed
Computed for Same 324
Observers

Evaluation: Optimal
Path from
Compressed Terrain
Tested on Original
Terrain Viewsheds –
14 of 4577 Points
[0.3%] Are
Erroneously Visible

W.Franklin, F.Luck, C.Wootert,
M.Jean, Z.Xu, D.Tracy, S.
Martin

2/16/2009
RPI Geo* Boothbay Harbor review
GeoStar Achievements

Dr W. Randolph Franklin
Assoc. Professor
Rensselaer Polytechnic Institute
ECSE Dept, 6003 JEC, 110 8th St, Troy NY 12180
wrf@ecse.rpi.edu, (518) 276-6077

Summary

This report describes how RPI has met its goals for the first phase of the GeoStar program. In addition to goals described in the contract proposal, we have met new goals presented at the periodic program review meetings.

We have produced two new lossy terrain representations, tiling and ODETLAP, which are both much more compact than gzip, the standard USGS terrain distribution format. We have evaluated our representations on more complex metrics that are closer to the end user. These new metrics include slope, optimal multiple observer joint viewshed, "smugglers" path planning to avoid those optimal viewsheds, and optimal path planning to fall within an given allowable maximum slope while minimizing material added and removed. We can also quickly classify points in order of importance on even 10000x10000 terrains, using our TIN program. Using ODETLAP, we can also fill in circles of missing data of diameter 200, while inferring local maxima. ESRI has written for us a DLL (i.e., extension) to ArcGIS, implementing our multi-observer siting algorithm.

The rest of this report gives more details, w/o repeating technical details that have already been presented several times.

- **Representations**
  - Regular tiling
  - ODETLAP
  - Triangulated irregular network
- **Software/algorithms**
  - Smugglers path planning
  - Path planning to minimize material moved
  - ESRI’s multiple observer siting toolkit software
  - Missing data fillin
  - Optional extension
  - Automated construction: proposed expanded task

http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Geo...
- Broader implications of this work for the military

Representations

Regular tiling

Our regular tiling representation partitions the terrain into 7x7 planar tiles, nonlinearly transforms the coefficients, and then encodes them with bzip2 and jpeg2000. We are still researching this representation, but it already obsoletes gzip for terrain. We evaluated regular tiling on the following six test datasets.

Test Data Sets

hill1

hill2
The following table evaluates regular tiling on them.
### Regular Tiling Results

<table>
<thead>
<tr>
<th>Data set</th>
<th>Orig. size (KB)</th>
<th>Gzip. size (KB)</th>
<th>Tiling size (KB)</th>
<th>Gzip/tiling</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>320</td>
<td>79</td>
<td>14.5</td>
<td>5.45</td>
<td>1.75</td>
</tr>
<tr>
<td>hill2</td>
<td>320</td>
<td>134</td>
<td>15.6</td>
<td>8.58</td>
<td>3.51</td>
</tr>
<tr>
<td>hill3</td>
<td>320</td>
<td>59</td>
<td>12.7</td>
<td>4.66</td>
<td>0.99</td>
</tr>
<tr>
<td>mtn1</td>
<td>320</td>
<td>177</td>
<td>15.7</td>
<td>11.3</td>
<td>7.92</td>
</tr>
<tr>
<td>mtn2</td>
<td>320</td>
<td>174</td>
<td>15.6</td>
<td>11.1</td>
<td>7.07</td>
</tr>
<tr>
<td>mtn3</td>
<td>320</td>
<td>174</td>
<td>15.7</td>
<td>11.1</td>
<td>7.04</td>
</tr>
</tbody>
</table>
The following scatterplot compares fractional error (i.e., RMS error divided by elevation range) and compression ratio relative to gzip.

We also evaluated the tiling representation using multiobserver visibility and path evaluation, as follows.

1. Site 64 observers on the terrain so as to maximize their joint viewshed. (The observer and target heights were chosen to be 10 and the viewsheds' radius of interest to be 30.)

2. Plan an optimal hidden ("smugglers") path between two distant points on the terrain, with the following cost metric. Traversing one hidden post cost one unit; traversing one visible post cost 100; traversing uphill cost more.

Note that our metric has local optima and the topological space of possible paths is not connected, both of which make the path planning harder.

3. Evaluate the tiling representation on the following two metrics.
   a. What is the area of the symmetric difference between the two joint viewsheds?
   b. When the path computed on the tiling representation is transferred to the original, more accurate representation, and the viewsheds and then the path's cost recomputed, how much does the cost increase? This is a measure of how bad was the path computation on the tiling representation.
The following table gives the results.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Viewshed error</th>
<th>Path error</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>0.54%</td>
<td>1.14%</td>
</tr>
<tr>
<td>hill2</td>
<td>1.49%</td>
<td>0.35%</td>
</tr>
<tr>
<td>hill3</td>
<td>0.05%</td>
<td>0.46%</td>
</tr>
<tr>
<td>mtn1</td>
<td>6.36%</td>
<td>12.1%</td>
</tr>
<tr>
<td>mtn2</td>
<td>6.94%</td>
<td>11.3%</td>
</tr>
<tr>
<td>mtn3</td>
<td>6.99%</td>
<td>17.6%</td>
</tr>
</tbody>
</table>

The mean compression ratio was 21.3 relative to the original binary format or 8.7 relative to gzip. The RMS elevation error was 4.6m. The mean viewshed error was 3.7%, and the mean optimal path error 7.2%.

**ODETLAP**

Our second alternate representation, ODETLAP, is more mature than tiling. ODETLAP is an extension of a Laplacian PDE to an overdetermined system of linear equations, which we solve to minimize RMS error. ODETLAP also supports observer siting and path planning quite well, in addition to elevation and slope.

The following figures show some of the computed paths when we evaluated our ODETLAP representation using the same joint viewshed and smugglers path metrics used for our tiling representation. (The top of each little lighthouse is the location of an observer.) Typically the viewshed error was about 1% and about 2% of the path computed on the ODETLAP representation was erroneously visible, as evaluated on the original representation.
We also evaluated the error in the terrain slope for the ODETLAP representation compared to the original representation. The user may trade off compression and accuracy. Here are some numbers for a particularly aggressive compression; remembering that the original binary files are 320KB.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>Elev Mean Abs Err</th>
<th>Elev Max Err</th>
<th>Slope Max Err</th>
<th>Slope Mean Abs Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mtn1</td>
<td>4206</td>
<td>11.541954</td>
<td>83.115980</td>
<td>52.5176</td>
<td>10.0689</td>
</tr>
<tr>
<td>Mtn2</td>
<td>4229</td>
<td>11.582295</td>
<td>67.538151</td>
<td>51.0946</td>
<td>9.9802</td>
</tr>
<tr>
<td>Mtn3</td>
<td>4251</td>
<td>10.723127</td>
<td>61.370492</td>
<td>59.5317</td>
<td>9.2252</td>
</tr>
<tr>
<td>Hill1</td>
<td>4071</td>
<td>2.450718</td>
<td>25.821692</td>
<td>19.2703</td>
<td>1.8377</td>
</tr>
<tr>
<td>Hill2</td>
<td>4099</td>
<td>6.366287</td>
<td>38.131314</td>
<td>33.6524</td>
<td>4.5922</td>
</tr>
<tr>
<td>Hill3</td>
<td>3902</td>
<td>1.260300</td>
<td>9.913579</td>
<td>13.4891</td>
<td>1.1336</td>
</tr>
</tbody>
</table>

**Triangulated irregular network**

We can compute a TIN representation of terrain on grids of 100,000,000 points on a laptop computer. In contrast to competing programs, we can process such datasets in memory without the need for paging the data in chunks from disk and processing it piecemeal. Our program is also incremental; it finds and inserts points in order of importance, rather than sequentially in order up the page. Thus, and unlike sweepline methods, it orders the points by importance, which is useful, e.g., for progressive transmission of the surface.

The following image shows a detail of a TIN representation of a surface that is two intersecting rings crossed by a road cut. We see the terrain ridges are automatically inserted into the representation without the need for any manual intervention.
Software/algorithms

Smugglers path planning

Answering a request last April, we have applied and extended the A* path planning algorithm. We apply path planning to evaluate alternate lossy representations as follows.

1. Plan a path on the alternate, lossy, representation.
2. Transfer that path to the original, assumed lossless, representation.
3. Evaluate the path’s quality on some metric.

For the smugglers metric, the path was planned on the alternate representation to avoid viewsheds that had been optimally sited to cover as much terrain as possible. Then the path was evaluated on how much was visible on the original representation.

Examples of this metric were shown above when discussing our alternate representations.

Path planning to minimize material moved

http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Geo...
This algorithm and program are designed to support automated road design. For each representation of the terrain, a road is constructed to minimize the amount of material added/removed, with the constraint that the slope never exceeds a given maximum. That was chosen to be the slope of a straight line connecting the two corners, which is actually quite strict. The A* algorithm is used. The volume of material added/removed in the alternate representation is compared against the volume added/removed when that same path is applied to the original representation. The percent difference between the two volumes is our error metric. In the table below, the alternate representation for each dataset was ODETLAP with 3000 points. Each dataset required about 6200 bytes, compared to the 320KB uncompressed binary representation. Our representation is lossy; note that that does not impede the path planning.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>% Difference in Volume Added or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1</td>
<td>0.084%</td>
</tr>
<tr>
<td>Hill2</td>
<td>1.536%</td>
</tr>
<tr>
<td>Hill3</td>
<td>0.093%</td>
</tr>
<tr>
<td>Mtn1</td>
<td>2.054%</td>
</tr>
<tr>
<td>Mtn2</td>
<td>0.004%</td>
</tr>
<tr>
<td>Mtn3</td>
<td>0.034%</td>
</tr>
</tbody>
</table>

The average material added or removed was 40 meters or more. Allowing greater slopes, or bridges and tunnels, would reduce this.

The following figures show four paths planned by our program on the datasets Hill2, Mtn1, Mtn2, and Mtn3. In each case, the path was planned from the top left to the bottom corner of the dataset.
Note how complex our paths are, and how they gradually climb up the terrain. Because they are usually not at the level of the terrain, because material is being added or removed during the optimization, this is again a more complex process than merely finding a locally optimal geodesic in a continuous scalar field.

**ESRI's multiple observer siting toolkit software**

ESRI has produced for us an ArcGIS DLL to compute multiobserver siting. The delivery consists of a CD-ROM containing the files for final delivery to produce the multiple observer siting toolkit software. It contains the source code, augmented with some testing and descriptive documents. The key deliverable is an operational class that can be configured to perform siting simulations on any platform with a C++ compiler. It also contains an ArcMap command application that runs on Windows, exercises the toolkit and demonstrates its capabilities. Both are functional and scalable. The results represent a common starting foundation for continued development; ESRI will continue developing this toolkit, as it shows promise for many applications. One need only load the Virtual C++ project and modify as desired.
One serendipitous application of our ODETLP representation is the ability to fill in large missing data.

http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Geo...
regions of missing data. Responding to a request at an early review meeting, the following figure shows six interpolation methods for filling in a missing hole of radius 100. The top left image, which is ODE TLAP, shows how local maxima in the missing region are inferred and realistic contours are generated. The top middle is a precursor to ODETLAP, and is slightly less realistic. The top right image is comparable to ODETLAP, but requires a higher order differential equation. The bottom three images show three Matlab techniques; all are quite unrealistic.

Optional extension

If our optional contract extension is granted we will, as described in the contract proposal, extend our representations to make them even more efficient, as well as extend our general scooping representation, which is still immature. We may also hire ESRI to extend its toolkit.

Automated construction: proposed expanded task
If more resources became available, we would like to extend our terrain research into the automated construction domain for roads, railroads, airfields, etc. In all of these, the goal is to site the facility where the construction will be cheapest, while maintaining other parameters, such as slope, within feasible ranges. One application would be the pacification of Afghanistan by means of increased linking infrastructure. Such automated construction is already a concern of offices such as Joint Rapid Airfield Construction (JRAC). We propose to increase productivity with Computational Geometry and Computer Science techniques.

The broader impact of this would be to do for construction what computer aided design has done for design over the last 30 years.

**Broader implications of this work for the military**

Our accomplishments; with devising and improving upon alternative representations of terrain using tiling, path planning and visibility hold broader promise for application in military engineering and battlespace environment technologies that support the Future Force. Automated site selection, assessment, and upgrading of existing airfields and other battlefield-relevant terraforms, like bunkers and defense positions are logical extensions to the algorithmic work we currently pursue. Site assessment, site feasibility, and design work can be more efficiently and usefully conducted remotely or with un-manned vehicles, as could battlespace terrain reasoning and awareness, and links with geospatial technologies. Extensions in these directions are consistent with Strategic Planning Guidance, the Army Science and Technology Master Plan (ASTMP), the Army Modernization Plan, and the Defense Technology Area Plan (DTAP). Their relevance to civil earthwork and construction application endeavors is also confirmed by interest expressed to us by NYDOT and Bentley Engineering Systems.


Page last modified on February 08, 2007, at 11:53 AM
Compact Visibility and Path Preserving Terrain Representations
DARPA/Geo* Review, Couer d’Alene

W. Randolph Franklin
Rensselaer Polytechnic Institute
Troy, NY, 12180
April 4, 2007

Outline I

1 Intro
   - Crew
   - Strategy
   - Test Data Sets

2 Achievements — Representations
   - Regular Tiling
   - TIN
   - ODETLAP
   - Segmentation
   - Polygonal Approximation
   - Scooping

3 Achievements — Operations
   - Multiple Observer Siting Toolkit
Outline II

- Path Planning for Road Construction
- Connected Components in Images

Future
- Combining Representations
- Globally Optimizing Representations and Operations
- Optimized Coding
- Drainage
- Scooping
- Productization

Summary

The Crew

- Dr W. Randolph Franklin – general secretary
- Dr Frank Luk – numerical computation (moving to HK in May)
- Dr Barb Cutler – computer graphics
- Dr Marcus Andrade – GIS, visiting from Brazil for a year (Dr Caroline Westort left in January).
- Metin Inanc – segmentation
- Zhongyi Xie – ODETLAP
- Dan Tracy – path planning
- Jon Muckell – connected components, drainage (another 2 players to be named in Sept)
Strategy

- Pursue several representations and applications in parallel.
- Following promising leads, synergize (e.g., ODETLAP is initialized from TIN), and productize (e.g., ArcGIS DLL for multiobserver siting)

Test Data Sets

- hill1 506m
- hill2 745m
- hill3 500m
- mtn1 1040m
- mtn2 953m
- mtn3 788m

- 400 × 400 points, 16 bpp, 320KB uncompressed binary size
- Number after each name is the elevation range
Regular Tiling
Method & Elevation Accuracy

- **Method**: Partition terrain into regular blocks, lossily compress with jpeg and bzip 2
- **Elevation Results**: (compared to gzip)

Regular Tiling
Viewshed and Path Planning Accuracy

- Optimally site observers on terrain.
- Optimally plan smugglers’ path avoiding observers.
- The optimal path minimizes this cost function:

\[
\begin{align*}
    c &= \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \\
    &\quad \cdot \left( 1 + \max \left( 0, \frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}} \right) \right) \\
    &\quad \cdot (1 + 100v)
\end{align*}
\]

- \( v = 1 \) iff we’re becoming visible to a guard.
- The 3 terms penalize long paths, traveling uphill, and being seen.
Regular Tiling
Viewshed and Path Planning Accuracy — Evaluation I

- Compute optimal path on compressed terrain.
- Project that path onto the original terrain.
- Compute the cost of the projected path and compare to the cost on the compressed terrain.
- If the cost is similar, then the compressed representation is adequate for the path planning task.

Regular Tiling
Viewshed and Path Planning Accuracy — Evaluation II

- The cost of the path evaluating on the compressed vs uncompressed terrain was very similar for the hill datasets, and differed by about 15% on the mountainous datasets.
- We will verify by comparing the cost of other paths on the terrain (e.g., optimizing different cost functions and also random paths).
Regular Tiling

Status

- Warmed us up on alternate representations.
- Served as a base for the segmentation representation below.
- Now serving as a testbed for coefficient coding techniques. PPMII (Prediction by Partial Matching with Information Inheritance) from LEDA (Library of Efficient Datatypes and Algorithms) may be the best.

TIN — Triangulated Irregular Network

- Can process $10^8$ points on a laptop.
- Works in memory w/o needing to page data from disk.
- Inserts points incrementally, in order of importance.
- Could be used progressively to transmit the surface.
- Identifies ridge lines automatically.
TIN — Triangulated Irregular Network

Impact

- Forms a basis for other techniques, such as ODETLAP, which process points in order of importance.
- Complements UNC’s TIN results.

ODETLAP – Overdetermined Laplacian

- Overdetermined linear system:
  - $z_{ij} = h_{ij}$ for known points,
  - $4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1}$ for all nonborder points.
- Fills contours to a grid.
- Fill missing data holes.
- Handles
  - incomplete contours,
  - complete contours,
  - isolated points,
  - inconsistent data.
ODETLAP – Overdetermined Laplacian

Evaluation Using Smugglers Paths

- Compute viewsheds and smugglers’ path on ODETLAP rep.
- Evaluate on original rep.
ODETLAP – Overdetermined Laplacian

- We were requested a year ago to upweight this.
- ODETLAP can fill in missing circles with \( r \leq 100 \).
- Slopes are continuous across the boundary.
- Contours are realistic.

6 fillin methods. Top left is ODETLAP. Top right is also good, but requires a higher order PDE. Bottom three are from Matlab.
ODETLAP – Overdetermined Laplacian
Missing Data Fillin — 2

RPI Geo* Final Report 582/919  Apr 2007 Review
Accurate representation of slope is important for mobility and drainage.

In these results, we reduced noise with a $3 \times 3$ convolution filter.
ODETLAP – Overdetermined Laplacian

Status

- Works well.
- Open avenues for future exploration.
  - Other PDEs
  - Other coefficient coders.
- Planning for possible productization by ESRI.

Segmentation

- **Description:** Divide the terrain into coherent pieces, which are easier to handle than the whole.
- **Challenge:** Sudden changes in source properties cause compression programs to subperform while they update their model.
- **Solution:** Partition a terrain into homogeneous, more easily compressed, pieces.
Segmentation
Algorithm

- Partition into $2 \times 2$ squares.
- Fit the best plane to each square.
- Expand each square into a segment containing all points with elevations within $\epsilon_{max}$ of the plane.
- From the large set of possible segments pick the smallest subset that covers the whole terrain.

Segmentation
Subtleties

- Level sets were inspirational.
- Use graph cuts to optimize choice of planar approximation (when more than one plane fits a data point within the acceptable error).
- Use exact intersection of planar (or non-planar) approximation surfaces to eliminate sharp vertical discontinuities when the approximating planar function changes.
- Also plan to use planar intersection remeshing code for that.
- Use shadow volumes for interactive visualization of complex terrain self-shadowing.
Polyhedral Approximation

**Intro**

- Lloyd's Relaxation ($k$-means clustering) to optimally select $k$ proxy planes
- Representation is simply the planes' *normal + centroid*
- To reconstruct, intersect the planes — Generalization of convex hull / half space intersection
- Advantages:
  - Connectivity is determined implicitly (does not need to be stored)
  - No artificial vertical discontinuities
  - Efficient rendering

---

Terrain Approximation

**Examples**

Elevs

Normals

200

1000

original

*Note: These are different data sets from our usual 6.*
Terrain Approximation

Error Map

200 planes: 150 × 150 × 30, Err: ±1.6 (5%)
1000 planes: Original, Err: ±0.8 (2.5%)

Scooping

Status

- This has been more challenging than expected.
- We are still researching the appropriate operators.
- To get concrete results quickly, we’ve been emphasizing other representations.
- Resources were also reallocated to path planning.
Multiple Observer Siting

- **Goal:** Greedily site hundreds of observers on terrain to maximize their joint viewsheds
- **Customizable:** Observer and target heights above local terrain, radius of interest
- ESRI has produced an ArcGIS DLL toolkit — an operational class configurable to perform siting simulations on any platform with a C++ compiler.
- Includes an ArcMap command application that runs on Windows, exercises the toolkit and demonstrates its capabilities.
- Both are functional and scalable.

Marquee Tool:

---

**Dialog Box:**
Multiple Observer Siting

ArcGIS DLL — 2

Original Data

Resultant Multiple Observer Siting Map

Siting and Path Planning

Site a group observers so as to maximize the amount of visible terrain. Use A* to plan a path through the terrain that avoids the visible areas while also minimizing the distance traveled. Use the visibility and the paths to evaluate terrain compression techniques.

Will help terrain to be stored more compactly and retrieved more quickly by soldiers in the field.

Future work: A toolkit for the path planning.
Path Planning

**Video**

RPI-path-planning.wmv

---

**Path Planning for Road Construction**

- **Goal:** Construct an optimal road connecting two points.
- **Allowed:** Material removal and deposition.
- **Constraint:** Max allowable slope is bounded.
- **Objective function:** Amount of material moved.
- **Method:** A*
Compute road (and its cost) on compressed rep, evaluate its cost on original rep, compare costs.

Restrict the several bulldozers to a maximum load per trip and a maximum traversal slope.
Connected Components in Images

- We can find the connected components in a 19000x19000 image, whose pixels have been classified, in 25 secs.
- This may be synergistically useful to other teams.
- Here is a detail of the output.

Future — Pruning & Combining Representations

- Combine representations to produce an even better rep.
- Note that the best text compression techniques do this.

**Example (jpeg)**
1. Rotate from RGB to YCrCb.
2. Perform a discrete cosine transform.
3. Perform a low-pass filter.
4. Arithmetic encode the remaining coefficients.

**Example (bzip2)**
1. Run length encoding.
2. Burrows-Wheeler transformation.
3. Move to front.
4. Another run length encoding.
5. Arithmetic encoding.
Allowing certain lossless operations to become lossy may make the global process more efficient, w/o losing global accuracy.

E.g., Could ODETLAP’s coding be more efficient if the \{(x, y)\} or \(z\) had errors? We would compensate by using more points.

The global error would remain the same.

This concept is potentially very powerful.

**Definition**

Coding is the process of transforming the coefficients into bits.

This is as important as determining the coefficients.
Drainage

- **Problem:** Local drainage regions trap water and inhibit long rivers.
- **Solution:** Use fast connected components.

Scooping

- Continue work on this representation, which has the most potential to:
  - model how much terrain is formed (by erosion),
  - represent discontinuities,
  - represent monotonic drainage nets.
Productization

- Hand off our most promising results to ESRI for productization.

Summary

Segmented Terrain

Smugglers Path

Original Surface
(320 KB)

Compressed Surface
(4071 Bytes)

Average Absolute Error = 2.451
Maximum Absolute Error = 25.822
Summary

Segmented Terrain

Smugglers Path

Summary

Segmented Terrain

Smugglers Path
RPI GeoStar Phase II

W. Randolph Franklin
Rensselaer Polytechnic Institute
ECSE Dept, 6003 JEC
110 8th St
Troy NY 12180

518 276 6077
frankwr@rpi.edu

This paper describes the tasks of Phase II of the RPI GeoStar project.

1. Task description

1. The first component is to extend Overdetermined Laplacian (ODETLAP), our compact terrain representation. ODETLAP has two steps:

   a) Determining a set of points from which we can reconstruct a good representation of the surface, when evaluated by some metric.

   The reconstruction is performed by setting up and solving an overdetermined linear system of equations. The unknowns are the elevations in a DEM. There is an equation setting each known point to its known elevation. There is also an equation for each nonborder point, known or not, setting it equal to the average of its four neighbors. The two types of equations can have different weights, depending on whether accuracy or smoothness is more important.

   b) Coding those points, that is, representing them in as few bits as possible.

   We have demonstrated ODETLAP enough that it is time to take it to the next level.

2. The second component is to extend Path Planning in a way that we call Smugglers and Border Guards. This component was not part of our proposal. However, in April 2006, Carey Schwartz asked us to study A* path planning. Initially we assumed that useful implementations were plentiful, but after not finding one, implemented our own. Two considerations were that we are in a Euclidean metric and that our datasets are larger (say 1201^2 points) than some A* algorithms are used to.
We build on our earlier multiple observer siting work by performing path planning on a terrain on which we have often first sited potentially hundreds of observers. The cost function has these components:

a) Minimize total path length,
b) Penalize uphill travel, and
c) Very strongly penalize traveling through any observer’s viewshed.

The following figure computed by one of our programs demonstrates a path that is longer but whose maximum slope is limited. (There are no observers to avoid in this version.) The path gradually climbs from the low region in the top left to the mountain top in the bottom right.

The following figure shows a path computed to avoid more than 100 viewsheds. (The source and goal were picked to make the path as complicated as possible.)
We are currently extending our path planning work to make the solution more realistic. Our paths are intended to represent real roads or railroads on terrain.

a) First, the path’s maximum slope may be limited. A railroad would have a lower limit than a road.

b) Second, since real roads and railroads are constructed with earth-moving equipment, we may modify the terrain so as to build a better road. In other words, earthmoving operations may be allowed. That is, we may produce a path that goes above or below the terrain. See the figure.

A hill and valley will make this road more expensive.

Level the hill; fill the valley.

Now, using the new road will be cheaper.

Now, the cost function has two components that trade off: the cost to build the road, and the cost to travel along it.

3. If there is interest, we can also extend our multiple observer siting algorithm. For example, instead of maximizing the visible area, perhaps it is more important that the
hidden area be composed of separate pieces, and not of one long region through which a smuggler might travel.

4. If there is time, I would like to return to scooping, as that representation has the greatest long-term potential. For the last year, the resources that would have gone here were redirected to path planning, which is also quite interesting.

2. What does the task accomplish?

This task accomplishes two different goals, one in terrain representation and the other in terrain operation.

1. First, it will produce a new more compact terrain representation, which also allows data sources with different accuracies and coverages to be conflated. That will allow larger volumes of terrain data to be used in the field, and allow base datasets to be augmented with data collected in the field.

2. Second, it will produce new algorithms and implementations for path planning on terrain while avoiding optimally sited observers. These paths will minimize a cost function including length, uphill travel, and time under observation. The construction of these paths may also allow earthmoving operations as described below.

3. Task approach

3.1 ODETLAP

1. We are investigating better coding of the \((x,y)\) coordinates, so as to approach the information theoretic bound for \((x,y)\). For 1000 points selected on a 1000\times1000 grid, that limit is 1.25KB for the \((x,y)\).

2. The information theoretic limit assumes no structure in the points. If there is structure, we might do better. In the limit, if the points form a regular grid, than coding the \((x,y)\) takes only the space to describe the grid. That problem with that is that the points are not adapted to the different complexity of various regions of the terrain. However that might be handled by a good compression algorithm for the \(z\). That is, if the \(z\)'s for adjacent points compress more when their values are close, then it is less important to adaptively select the points.

3. An intermediate approach might be to have overlapping local regular grids of points. This is reminiscent of overlapping local coordinate systems in differential geometry.

4. We might select points to optimize more sophisticated metrics, such as
   a) slope,
   b) visibility (i.e., select some highly visible points as well as points with large errors.
5. To date, detailed point selection strategies haven’t seemed to make a big difference. (We investigated selecting points with the largest error, points that were most visible, and other variations.) Therefore, instead, can we select points that compress more? For example, this sequence of z: (1, 2, 3, 4, 5, 6, 7) will compress more than this sequence: (8, 3, 5, 1, 7, 4, 4). Our goal is not to minimize the number of points needed to represent the terrain to some given accuracy, but rather to minimize the number of bytes needed. If we use more points but they compress better, then that is a win.

To illustrate that idea, here’s something that we tried that didn’t work. If we force the points’ x and y coordinates to be even, then they will compress better by one bit per coordinate. However, since they are then less accurate, more will be needed. This particular tradeoff failed, but the principle looks promising.

6. It is also appropriate now to add a hierarchy to ODETLAP. One approach would be as follows.

   a) Initially, coarsen the point grid, say by selecting every k-th row and column.

   b) Run ODETLAP on that, then move to the full dataset.

7. Another approach is as follows.

   a) Initially run ODETLAP with a larger R (more smoothness and less accuracy).

   b) Find the error matrix.

   c) Run ODETLAP with a smaller R on that.

   d) Repeat.

8. The z values don’t compress much (because the points have been selected to be nonredundant). We are studying the following promising approach.

   a) Once the points have been selected, compute a minimal traveling salesman path on them.

   b) Produce a list of z’s in that order. Adjacent z’s should have similar values since their points are close.

   c) Now, compress them, initially with delta encoding, later with more sophisticated methods.

9. Another enhancement to ODETLAP is **terrain segmentation**, to partition the cell into regions where the terrain is internally consistent. For example a mountain arising from a plane would generate two segments. The thesis is that each segment would compress better with ODETLAP. The following figure shows the segments computed in the current version of our program. The segmentation is a generalization of level sets. Each segment can be approximated by an inclined plane. A segment need not be a connected set. A segment is grown from a seed by including all points close enough to the inclined plane. The major possible
tradeoff is that a more complicated segment takes more information to describe its boundary but less information to describe the points inside it.

10. During this work, we are also evaluating ODETLP on accuracy of slope reconstruction. Currently, when our six 400x400 datasets are compressed to about 4K, or 0.2 bits per point, the average slope error ranges from 1% to 10%, with the less mountainous terrains being represented more accurately.

11. We are also solving details to to make ODETLP more useful in a production environment, such as the following.

   a) The ODETLP equations for border points of the cell have not yet been completely resolved. Some issues are that certain equations may bias the surface to be horizontal at the border, while certain other equations may be redundant, leading the system to become underdetermined at times. This is similar to setting the end condition for a spline curve. We are adapting spline ideas here.

   b) To date, we have been running ODETLP in Matlab on 400x400 cells. Sometime, we should transition to processing full 1201x1201 cells, which will probably require leaving Matlab. John Childs did this as a masters project for Franklin a few years ago. He used a Saunders-Paige technique. We might adapt and integrate his ideas.

3.2 Path planning
1. Instead of volume of material added/removed as the cost metric for the road construction, assume that the material is being moved by some machine, such as a bulldozer, with a finite capacity per load, and use the total distance that it travels.

2. The bulldozer cannot travel across too steep a slope. The bulldozer thus may not always be able to simply dump material at the closest off-road location. It may even have to construct a slope just to be able to reach the road construction site, or at least to shorten the path.

3. In addition, our current path planning code is preliminary and special-case, and needs refactoring to make it generally useful and faster.

4. Task advantages

1. More compact terrain representations are required by the ever larger amounts of terrain data. ODETLAP is designed to require more resources to encode the terrain into that representation than to decode it back to a DEM. That is appropriate since a cell will be decoded more often, and on smaller computers, than it will be encoded.

2. ODETLAP can conflate overlapping inconsistent cells of terrain data. For example, the input might be a large, low precision cell that is overlaid by various smaller high precision cells. This ability arises naturally from the fact that ODETLAP is solving an overdetermined, that is, inconsistent linear system. We are unaware of any other terrain representation that can easily do this.

3. Our path planning work is of military importance because it allows efficient routing to be performed by nonspecialists. There are diverse applications. For example, the observers might be antiaircraft batteries, and the paths to be flown by our aircraft.

4. The earthmoving aspect of our path planning is extending the idea of "know the earth" to ".. and then optimize it".
4.1 Summary for NGA – Aug 2007

Objectives

This summary of the project was prepared at NGA’s request.

1. ODETLAP, a new compact terrain representation that supports multiple observer siting and path planning
2. Siting of hundreds of observers to maximize their joint viewshed
3. Optimal path planning to avoid the observers’ viewsheds

Key Differentiating Factors

1. a smooth terrain representation, w/o visible blockiness.
2. the ability for progressive transmission of the terrain; the longer the transmission, the more accurate is the reconstructed surface.
3. the ability to conflate inconsistent partially overlapping data sets.
4. the ability to interpolate partial sets of elevation posts, generating continuous slopes even when the input data consists of nested contour lines.
5. the ability to infer local maxima inside the topmost contour.
6. a procedure to site hundreds of observers on the terrain. The observers’ radius of interest may be several hundred pixels.
7. a visibility and navigation tool, Smugglers and Border Guards, that plans a path by optimizing a sophisticated objective function consisting of the path’s total distance, the amount of uphill travel, and the distance spent in sight of any observer. The paths may be thousands of pixels long.
8. Finally, Smugglers and Border Guards is used to evaluate ODETLAP. That is our alternate terrain representation not only produces surfaces with a good RMS error but surfaces on which observers can be accurately sited and paths accurately planned.

Potential Benefits

1. More compact terrain representations are required by the ever larger amounts of terrain data. ODETLAP is designed to require more resources to encode the terrain into that representation than to decode it back to a DEM. That is appropriate since a cell will be decoded more often, and on smaller computers, than it will be encoded.
2. ODETLAP’s ability to conflate overlapping inconsistent cells of terrain data allows a large, low precision cell to be overlaid by various smaller high precision cells, producing one unified elevation field.
3. Our path planning work is of military importance because it allows efficient routing to be performed by nonspecialists. There are diverse applications. For example, the observers might be antiaircraft batteries, and the paths to be flown by our aircraft.
DEM Compression and
Terrain Approximation;
Smugglers and Border
Guards

Prof W Randolph Franklin, Prof Frank Luk, Prof
Barbara Cutler, Prof Marcus Andrade, Metin
Inanc, Zhongyi Xie, Dan Tracy, Jon Muckell
Rensselaer Polytechnic Institute
mail@wrf franklin.org, 703-447-7808

Complete Record

- This version contains both the slides that I showed
to NGA (on 10/1/2007), and to DARPA (on
10/2/2007). Therefore it is both a good
introductory and detailed record of RPI’s
performance to date.
- http://www.ecse.rpi.edu/~wrf/pm wiki/GeoStar
contains most material ever given by RPI to
DARPA or NGA.

Ask WRF for a password
Team

- Prof Randolph Franklin – helping everyone
- Prof Barbara Cutler – computer graphics
- Prof Frank Luk (on leave as Vice-President (Academic) of Hong Kong Baptist U) – numerical analysis
- Prof Marcus Andrade – visiting from UF Viçosa (Brazil) – computational geometry
- Metin Inanc – ODETLAP
- Zhongyi Xie – ODETLAP
- Dan Tracy – multiobserver siting, path planning
- Jon Muckell – hydrology.

Potential Benefits

- More compact terrain representations.
  - Store the ever larger amounts of terrain data. Spend time on the compression (which is done once) to get most compact representation.
  - Works on 16 bit topography.

- Conflate overlapping inconsistent cells.
  - Overlay large, low precision cell by smaller high precision cells => one unified elevation field.

- Efficient routing by nonspecialists.
  - Route our aircraft away from antiaircraft batteries

- Better hydrographic computations.
  - Flood prediction.
An Inadequate Terrain Representation
Goal 1: DEM Compression and Terrain Approximation

- ODETLAP: alternate terrain representation.
- Compact.
- Allows lossy - size / quality tradeoffs.
- Emphasized decompression speed.
- Evaluated on visibility, mobility metrics.

Milestone Progress

- Phase I: 10x compression while maintaining usefulness; Phase II: 100x
- Reverse engineered HRTI Analysis Tool's slope formula to avoid running HAT each time.
- We gave before-and-after data to NGA demonstrating this.
- Further improvements made since then.
- Fitting other components of proposal (e.g., hydrology) to correspond to milestones.
Key Differentiating Factors

- Smooth representation
  - no visible blockiness
- Allows progressive transmission
  - longer transmission => more accurate reconstruction
- Conflates inconsistent partially overlapping data.
- Interpolates partial sets of elevation posts
  - generates continuous slopes even when the input data consists of nested contour lines.
- Infers local maxima inside the topmost contour.

ODETLAP Process

Since our first description of ODETLAP at the 1998 Spatial Data Handling Symposium, we’ve built this system.
### ODETLAP Point Selection

**Several options:**

- Incremental TIN to find most important points, then greedy insertion of worst points *(Allows progressive transmission)*
- Regular grid of points *(more points, but compress better)* *(More compact)* **NEW**
- Stream and ridgeline points *(Preliminary)* **NEW**

### Other Point Selection Strategies

The following were not as good:

- Select highly visible points, **or**
- Random points, **or**
- Points based on histogram of heights with boosted sampling for less frequent elevation bands and small connected components.
Incremental Triangulated Irregular Network (TIN)

- Can process $10^8$ points on a laptop.
- Works in memory w/o needing to page data from disk.
- Inserts points incrementally, in order of importance.
- Can progressively transmit terrain.
- Identifies ridge lines automatically.

Coding the Points to Reduce Space

- Code $(x,y)$ separately from $z$. **NEW**
- If $(x,y)$ a regular grid: give its resolution
- Else: run-length encode the bitmap.
  - 0100000011000010001 -> 16043
  - Only about 1000 of 160,000 bits are 1.
- $Z$: delta code, then $bzip2$.
  - 100 125 90 90 100 -> 100 25 -35 0 10
Traveling Salesman Path

- **NEW!**
- Hypothesis: nearby points often have nearby Z, which delta code better
- Find a traveling salesman path through the selected points.
- Put the Z in that order and code them.
- (X,Y) coding is not affected.
- **Status:** have some preliminary results.

Info theoretic min for (x,y)

- Assume that 1000 of 400x400 bits are 1, rest are 0.
- Assuming no structure in the 1s, size is
  \[ \lg(\text{choose}(160000,1000)) = 8754 \text{ bits} \]
- We approach that within 20%.
- That’s why we separate (x,y) from (z).
Better Than the Info-Theor Limit?

- The information theoretic limit was calculated assuming no structure in the points.
- Is there a structure to exploit?
  - Scooping
  - Grids of points

Reconstruction Context

- Extension of classical Laplacian partial differential equation used to solve heat flow etc
- Now possible with new numerical computation techniques on large sparse overdetermined systems of linear equations
- Adds capabilities to the classical system
  - Local maxima inference
  - Inconsistent data conflation
**ODETLAP Point Reconstruction**

- Solve an overdetermined variant of a Laplacian PDE.
  - Known pts: \( z_{ij} = h_{ij} \)
  - All pts: \( 4z_{ij} = z_{i-1j} + z_{i+1j} + z_{ij-1} + z_{ij+1} \)
- Easily processes 400x400 arrays of elevation posts in Matlab (160,000 unknowns)
- Process larger arrays with Page-Saunders technique

**ODETLAP on Larger Cells**

- We could go to Page-Saunders if there is interest.
- My masters student John Childs did this in 2003, before Geo*.
- **Goal:** several-thousand-square cell.
Four Matlab Interpolation Techniques on Nested Square Contours

This difficult example was chosen to illustrate all these methods’ limitations.

ODETLAP on Nested Squares

Surface now looks much better. Can tradeoff accuracy vs smoothness.
Terrain Test Data

Extracted from level 2 DEMs

Hill1

Hill1

Elevation range

mtn1 1040  mtn2 953  mtn3 788

hll1 505  hll2 745  hll3 500
Accuracy Metrics

- Since flatter cells are easier,
- Following slides and tables show
  - RMS error, meters, or
  - (RMS error) / (elevation range in the cell)
- Slope computed using Zevenbergen-Thorne algorithm used in NGA HAT.
- Slope error is always RMS degrees.
### TIN + Greedy ODETLAP Results

<table>
<thead>
<tr>
<th>Data</th>
<th>Size, bytes</th>
<th>Compression ratio</th>
<th>RMS Elev Error, m</th>
<th>RMS Slope Error, deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>1880</td>
<td>170:1</td>
<td>2.83</td>
<td>3.53</td>
</tr>
<tr>
<td>hill2</td>
<td>1962</td>
<td>163:1</td>
<td>4.06</td>
<td>8.06</td>
</tr>
<tr>
<td>hill3</td>
<td>1739</td>
<td>184:1</td>
<td>1.66</td>
<td>1.65</td>
</tr>
<tr>
<td>mtn1</td>
<td>1979</td>
<td>162:1</td>
<td>3.77</td>
<td>14.0</td>
</tr>
<tr>
<td>mtn2</td>
<td>2006</td>
<td>160:1</td>
<td>4.31</td>
<td>14.1</td>
</tr>
<tr>
<td>mtn3</td>
<td>2004</td>
<td>160:1</td>
<td>4.58</td>
<td>13.3</td>
</tr>
</tbody>
</table>

### TIN+Greedy Elevation Accuracy

![Graph showing compressed size vs. error for different data sets](image)

- **Hill 1**
- **Hill 2**
- **Hill 3**
- **Mtn 1**
- **Mtn 2**
- **Mtn 3**

The graph illustrates the relationship between compressed size and RMS elevation error for various datasets.
TIN+Greedy Slope Accuracy

Compressed Size vs. Error

TIN+Greedy Elevation Comparison
Mtn2 Dataset

7641 bytes => 42:1 compression ratio
**Zevenbergen-Thorne Slope Comparison**

**Mtn2 Dataset**

![Slope Comparison Diagram](image)

- mtn2: Original Slope
- mtn2: Reconstructed Slope

Zevenbergen-Thorne method; Slope in (degrees)

7641 bytes => 42:1 compression ratio

---

**Even Better Slope Representation**

- **Idea:** Extend the ODETLAP equations directly to incorporate the original representation’s vector gradient at critical points, instead of inferring slope from adjacent elevations.
- **Status:** being designed.
Different Point Selection Strategies

- Previous slides used TIN+greedy
- That can be made to allow progressive transmission of the points, by replacing bitmap coding of the (X,Y) with a bzip2 compression.
- Following slides use regular grid point selection.
- That compresses better but doesn’t do progressive transmission.

Regular Grid ODETLAP Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Points</th>
<th>Compressed Size</th>
<th>Compression Ratio</th>
<th>Elev RMS (m)</th>
<th>Slope RMS (deg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1</td>
<td>529</td>
<td>306</td>
<td>1046:1</td>
<td>9.63</td>
<td>4.32</td>
</tr>
<tr>
<td>Hill2</td>
<td>1600</td>
<td>807</td>
<td>397:1</td>
<td>9.98</td>
<td>6.54</td>
</tr>
<tr>
<td>Hill3</td>
<td>225</td>
<td>172</td>
<td>1860:1</td>
<td>9.71</td>
<td>3.04</td>
</tr>
<tr>
<td>Mtn1</td>
<td>4489</td>
<td>2194</td>
<td>146:1</td>
<td>9.66</td>
<td>10.13</td>
</tr>
<tr>
<td>Mtn2</td>
<td>4489</td>
<td>2027</td>
<td>158:1</td>
<td>9.95</td>
<td>10.34</td>
</tr>
<tr>
<td>Mtn3</td>
<td>4489</td>
<td>2013</td>
<td>159:1</td>
<td>9.91</td>
<td>9.85</td>
</tr>
</tbody>
</table>

(Our second ODETLAP point insertion strategy)
Regular Grid ODETLAP Accuracy

Merged Point Selection Strategy

- Start with regular grid
- Greedily add more points,
- Use overlapping local grids, like with differential geometry coordinate frames.
- Status: being designed.
Missing Data Fillin

- ODETLAP can fill in missing circles with \( r \leq 100 \).
- Slopes are continuous across the boundary.
- Contours are realistic.
- Next slide compares ODETLAP to 3 Matlab methods.

Fillin Comparison

<table>
<thead>
<tr>
<th>ODETLAP</th>
<th>Laplacian</th>
<th>Thin plate</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="ODETLAP" /></td>
<td><img src="image" alt="Laplacian" /></td>
<td><img src="image" alt="Thin plate" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matlab nearest</th>
<th>Matlab linear</th>
<th>Matlab cubic</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Matlab nearest" /></td>
<td><img src="image" alt="Matlab linear" /></td>
<td><img src="image" alt="Matlab cubic" /></td>
</tr>
</tbody>
</table>
Packaging ODETLAP

- Current process is a group of programs combining Matlab, bzip2, C++.
- Being packaged into a unified system for distribution to NGA.
- We can complete this when needed.

More Terrain Representations

- Scooping still has the greatest longterm potential.
- Could use RPI’s BlueGene/L, the 2nd fastest computer in a university setting in the world. (source: http://top500.org/lists/2007/06)
- Problem: It doesn’t run Matlab.
**Goal 2: Smugglers and Border Guards (aka Siting & Path Planning)**

- **Terrain**
- **Parameters:**
  - Observer height
  - Target height
  - Radius of interest
  - Intervisibility?
- **Siting program**
- **Observer positions**
- **Joint viewshed**

---

**Multiobserver siting steps**

1. Compute approximate visibility index of every possible observer.
2. Compute exact viewsheds of the best.
3. Greedily insert potential observers into the final set of observers, maintaining a bitmap of the cumulative viewshed.
4. Intervisibility => insert only visible observers.

**Key:** fast bitmap operations allow hundreds of observers to be sited with hi-res viewsheds.
Sample Viewsheds

Note the level of detail

Viewshed uncertainty

Hue indicates elevation

Visible
Possibly hidden
Hidden

Observer

Probably hidden
With or w/o intervisibility

Intervisibility enforced

No intervisibility required

Color -> elevation; Black -> hidden.

Siting Toolkit by ESRI

- **ArcGIS DLL** toolkit: an operational class configurable to perform siting simulations on any platform with a C++ compiler.
- Includes an ArcMap command application on Windows, to demonstrate its capabilities.
- Both are functional and scalable.
- Marquee Tool:
**ArcGIS DLL Dialog Box**

![ArcGIS DLL Dialog Box Image]

**Path Planning (Smugglers)**

- Find cheapest path between source and goal.
- Cost metric is not simply path length:

\[
\begin{align*}
    c &= \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \\
    &= \left(1 + \max \left(0, \frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}}\right)\right) \cdot (1 + 100\nu) \\
    &\quad \text{(Distance)} \\
    &= \left(1 + \max \left(0, \frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}}\right)\right) \\
    &\quad \text{(Climbing costs)} \\
    &= \left(1 + \max \left(0, \frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}}\right)\right) \\
    &\quad \text{(BIG penalty for being seen)}
\end{align*}
\]
Path planning algorithm

- Designed for hi-res, say 1000x1000, maps.
- Impossible to form the $10^6 \times 10^6$ cost matrix.
- Use A* to search for initial feasible, good, path.
- Iterate to optimize it.
- Doesn’t hang up on local optima.
- Compute many paths to evaluate compression throughout the terrain.
- Note how complex our paths are.
- Video: multipath.wmv

Many Paths on Each Dataset

hill1  hill2  hill3

mtn1  mtn2  mtn3
Many Paths on Hill1

Computed between 50 pairs of random start/end points

Many Paths on Mtn1
Many Paths on Mtn3

Path traversal video
- RPI-path-planning.wmv
**Alternate Terrain Representation Evaluation Using Path Planning**

- **Q:** Our alternate compressed representation has a good RMS elevation error. However, is it good for more sophisticated operations, like path planning?
- **If we compute a smugglers path on terrain stored in our alternate rep, how good is it really?**
- **It's not important if a computed path is very different from the optimal path, provided that its true cost is not much more expensive than the optimal path.**

---

**Smugglers Path Evaluation of ODETLAP**

- **Size:** size of compressed dataset in bytes. Original binary size = 320KB.
- **Incr. cost:** extra cost of optimal path computed on compressed dataset and evaluated on original dataset compared to optimal path computed on original dataset.

<table>
<thead>
<tr>
<th>Data</th>
<th>Size</th>
<th>Compr. Ratio</th>
<th>Incr. Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>1763</td>
<td>182:1</td>
<td>5.5%</td>
</tr>
<tr>
<td>hill2</td>
<td>1819</td>
<td>176:1</td>
<td>6.1%</td>
</tr>
<tr>
<td>hill3</td>
<td>1607</td>
<td>199:1</td>
<td>4.4%</td>
</tr>
<tr>
<td>mtn1</td>
<td>1925</td>
<td>166:1</td>
<td>19.2%</td>
</tr>
<tr>
<td>mtn2</td>
<td>1884</td>
<td>170:1</td>
<td>18.2%</td>
</tr>
<tr>
<td>mtn3</td>
<td>1946</td>
<td>164:1</td>
<td>17.0%</td>
</tr>
</tbody>
</table>
Path Planning for Road Construction

- **Goal:** Construct an optimal road connecting two points.
- **Allowed:** Material removal and deposition.
- **Constraint:** Max allowable slope is bounded.
- **Objective function:** Amount of material moved.
- **Method:** A*

Several Computed Roads
Key Differentiating Factors – Smugglers and Border Guards

- Can site hundreds of observers on the terrain.
- Observers’ radius of interest may be several hundred pixels.
- Visibility and navigation tool
  - optimizes a sophisticated objective function consisting of the path’s total distance, the amount of uphill travel, and the distance spent in sight of any observer.
- Paths may be thousands of pixels long.
- Validates ODETLAP:
  - Good RMS elevation error but wait, there’s more!
  - Can be used accurately to site observers and plan paths.

Hydrology Problem

Just starting this Phase II task in the proposal.

- Compute streams from terrain, ...
- Assuming water flows from each cell to lower neighbors.
- Problem: many cells are local minima trapping flow
- Why? Data errors; insufficient sampling
- Effect: No long streams.
Common Solution

- Simulate gradually filling in the local minima until the water flows over the edge.
- *But:* This is slow

- Real-world implementation of this technique with the Taum Sauk reservoir; before and after.

Our Techniques

- Identify sinks & watershed boundaries with our very fast connected components program.
- Solve for water flow as a sparse system of linear equations.
- Invert elevations and solve for *ridge rivers*.
- Merge watersheds not blocked by significant ridges.
- Input ridge and stream points to ODETLAP.
Fast Connected Components

- **Original application:** Small block of concrete is stressed to failure while being CAT-scanned in Brookhaven synchrotron. First breaks into a few large blocks, ...
- Need the 3D structure to understand failure.
- Compute the connected components of the thresholded 1000x1000x1000 CAT scan.
- **Next slide:** slices in 3 different directions look quite different.

Slices in 3 Directions of Cracking Concrete Block
Method

- Good implementation of union-find.
- Fundamental data structure is a 1-D run of solid voxels. *(assumes data coherence)*
- Carefully designed, small and fast.
- Form connected components with a few passes through the data.

Largest Test Run

- **Input:** 1024x1088x1088 = 1,212,153,856 voxels, 50% empty.
- 20,216,828 1-D runs, averaging 30 voxels.
- **Output:** 4,539,562 components, averaging 4.5 runs.
- **Largest component:** 2993 runs, volume 23675.
- **Many components:** only one run and two voxels.
- **Implementation:** 2GHz IBM t43p laptop, linux, gcc.
- **Virtual memory used:** only 340MB
- **Elapsed time:** 51 CPU seconds.
- **Times scale down:** ≈ 0.1 secs for 100x100x100.
- **Largest 2D test:** 19000x19000
Merging Watersheds not Blocked by Significant Ridges

Before merging

After merging

=> Larger more realistic watersheds and drainage networks
Summary

- Represent terrain in 1% of original binary space with compression ratios of 80:1 to 500:1 with 10m elevation and 5-10 degree slope error.
- Site multiple observers ("border guards") and then plot smugglers paths to avoid them.
- Compute on the compressed terrain.
- Modify terrain to improve hydrology.
Note to Carey Schwartz on 19 Nov 2008 responding to his change in our research direction

We're glad to show how well we can handle this major change in our original mission, downgrading the importance of aggressive elevation compression. Since your previous major suggestion, to do path planning, led us into such a fruitful area, we are excited about this one. The closer we are to the users, the more useful we can be.

I started thinking of strategies as soon as the St Louis NGA folks introduced the 2-degree slope criterion to me in October, and described some preliminary ideas to Ed Bosch then. Since that meeting, my students have tried some different point selection methods that had the goal of pinning down the slope. The hope was that if we specified several close points, we'd by implication specify the slope there. The problem is that slope's autocorrelation distance is so small that very many points are required.

We're now trying an overdetermined version of a thin plate PDE instead of the Laplacian PDE. However, we'll probably need to compute the slope explicitly with the Z-T method, then compress it, and finally use it to perturb the compressed elevation, to produce a final restored elevation matrix whose slope is as desired. This should work because representing slope at a point to, say 1 degree, requires many fewer bits than representing elevation at that point. In other words, our alternate representation will still be compact, just not quite as small.

This is an example of a deeper problem that was perhaps mentioned in my original proposal. That is, how to compress correlated layers of data while preserving the relations. So, we might be able to generalize our solution.

This bears some similarity to how Stan Osher does slopes. However, since we are able to route much more complicated paths than he, we hope also to be able to do slopes even better.

(Our path planning is better because we handle complicated topologies. That is, the hardest part is determining on which side of each observer to put the path. It's not a simple matter of locally optimizing a geodesic. Also, our paths are much longer.)

A flow diagram of our proposed approach is attached. The difficult part is the merging step. We plan to use an overdetermined sparse linear system, since that technique has worked so well. What the particular equations will be is still be considered.

We we really need is data, and data that can be given to aliens. NGA had said they'd get us some; we're still waiting.

Also, not having to productize our work for NGA frees up resources. Your encouragement at the Oct meeting to do our own deals with companies was welcome. I am considering how to transition this work myself after our DARPA award is over. A commercial partner would want to work with me (instead of, say, merely working off of our reports) because I understand this work better than anyone else, since I originated basically all the ideas and also wrote the first version of most of the software.
Page last modified on February 26, 2008, at 04:41 PM
DARPA/ DSO Geo* project at RPI (PI: W. Randolph Franklin)

GeoStar

Feb 2008

RPI GeoStar Task Summary

W. Randolph Franklin, PhD
Rensselaer Polytechnic Institute
ECSE Dept, 110 8th St, Troy, NY, 12180
frankwr@rpi.edu, 518-276-6077
Feb 2008

This is a report on the DARPA/DSO GeoStar award to Rensselaer Polytechnic Institute, detailing the assigned tasks, accomplishments, deliverables, and what remains to do.

To people reading a paper or PDF version of this report: the original is online in the wiki containing all our reports at http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/GeoStar/GeoStar. Please ask for the passwords.

1. Tasks
   1.1. Proposed in contract
   1.2. Listed in NGA summary slide
   1.3. Post-award modifications
2. Tasks accomplished
   2.1. Terrain representation - Morphological terrain sculpting
   2.2. Terrain representation - overdetermined Laplacian PDE (ODETLAP)
   2.3. Terrain operators - Siting/intervisibility toolkit
   2.4. Terrain representation - TINs
   2.5. Terrain representation - lossy compression by a factor of 100
   2.6. Path Planning
3. Tasks to be completed
   3.1. Slope
   3.2. Terrain representation - drainage analysis
   3.3. Code Cleanup
4. Deliverables
   4.1. TIN Code
   4.2. ODETLAP Code
   4.3. Siting Code
   4.4. Path planning to minimize material moved
   4.5. Papers
5. Corrections

1. Tasks
The tasks assigned to RPI in the original proposal have been updated with numerous modifications. These provided us with the opportunity to develop useful ideas and tools that had not occurred to us during the proposal preparation. The details are as follows.

1.1. Proposed in contract
The following tasks were listed in our proposal. For reference, that is here.

1. Terrain representation - Morphological terrain sculpting
2. Terrain representation - overdetermined Laplacian PDE (ODETLAP)
3. Terrain operators - Siting/intervisibility toolkit
4. Terrain representation - TINs
5. Terrain representation - lossy compression
6. Terrain representation - trajectory planning
7. Terrain representation - drainage analysis

1.2. Listed in NGA summary slide
Of the tasks that we proposed, the NGA summary slide presented at several review meetings assigned RPI this goal:

1. In phase I, compress terrain by a factor of 10.
2. In phase II, compress terrain by a factor of 100

1.3. Post-award modifications
At various times after the contract was awarded, Carey Schwartz, our DARPA POC, requested the following modifications, to better support NGA.

1. Make path planning an important theme. (This request was made at the April 2006 Snoqualmie review.)
2. Spend less time on productizing our work.
3. Pick our best terrain representation and concentrate on it, instead of spreading our efforts.
4. In particular do not award most of the money to ESRI that had been budgeted in the proposal.
5. Stop concentrating on extreme terrain compression. Rather, concentrate on representing slope accurately. This major change was requested on 11/7/2007:

"I'm asking the agent to redirect a portion of your research work to focus on compressing hi-res urban LIDAR data rather than the level II DTEDs, to focus on extreme fidelity rather than extreme compression in order to achieve a 2 degree threshold for slope accuracy and that you
focus on improving the results from his software before putting too much effort into productizing your SW."

In response to these requests, we de-emphasized and then largely stopped morphological and scooping terrain representation. Instead, we concentrated on our ODETLAP terrain representation. We believe that scooping has the greatest long-term potential for a revolutionary new representation. To facilitate that work, we had hired Dr Caroline Westort as a Research Assistant Professor from about Aug 2005 to Dec 2006, using money that would have gone to ESRI. However, ODETLAP has the greater short-term potential. At this point, ODETLAP has become a base for a class of related techniques.

2. Tasks accomplished

We have accomplished the following tasks. More detail was presented at the NGA Industry Day and DARPA review day on Oct 1 and 2, 2007. A merged presentation is RPI-Oct2007-merged.ppt.

This list is presented in the same order as the tasks listed above.

2.1. Terrain representation - Morphological terrain sculpting

As described above, although this has the greatest long-term potential, we have transferred the resources originally devoted to this to the newer assigned tasks, in order to produce more results faster.

2.2. Terrain representation - overdetermined Laplacian PDE (ODETLAP)

We have developed this overdetermined extension to the Laplacian partial differential equation for terrain representation. It has the following key differentiating factors:

1. a smooth terrain representation, w/o visible blockiness.

2. the ability for progressive transmission of the terrain; the longer the transmission, the more accurate is the reconstructed surface.

3. the ability to conflate inconsistent partially overlapping data sets. That is, it can overlay a large, low precision cell by smaller high precision cells, and create one unified elevation field.

4. the ability to interpolate partial sets of elevation posts, generating continuous slopes even when the input data consists of nested contour lines.

5. the ability to infer local maxima inside the topmost contour.

This is the development that we used to compress the terrain by a factor of 100.

http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Ge...
2.3. Terrain operators - Siting/intervisibility toolkit

ESRI has produced for us an ArcGIS DLL to compute multiobserver siting. The delivery consists of a CD-ROM containing the files for final delivery to produce the multiple observer siting toolkit software. It contains the source code, augmented with some testing and descriptive documents. The key deliverable is an operational class that can be configured to perform siting simulations on any platform with a C++ compiler. It also contains an ArcMap command application that runs on Windows, exercises the toolkit and demonstrates its capabilities. Both are functional and scalable. The results represent a common starting foundation for continued development; ESRI will continue developing this toolkit, as it shows promise for many applications. One need only load the Virtual C++ project and modify as desired.

Marquee Tool

Dialog Box
2.4. Terrain representation - TINs

We can compute a TIN representation of terrain on grids of 100,000,000 points on a laptop computer. In contrast to competing programs, we can process such datasets in memory w/o the need for paging the data in in chunks from disk and processing it piecemeal. Our program is also incremental; it finds and inserts points in order of importance, rather than sequentially in order up the page. Thus, and unlike the competing sweepline methods, it orders the points by importance, which is useful, e.g., for progressive transmission of the surface.

The following image shows a detail of a TIN representation of a surface that is two intersecting rings crossed by a road cut. We see the terrain ridges are automatically inserted into the representation w/o the need for any manual intervention.
2.5. Terrain representation - lossy compression by a factor of 100

We can compress terrain by a factor of 100 compared to the binary representation of the elevation data, while maintaining a reasonable error.

The purpose is to store the ever larger amounts of terrain data. Our strategy is to spend time on the compression (which is done once) to get the most compact representation. We process 16 bit topography (unlike some other methods).

The following table shows our results on six sample level-2 DEM datasets.

<table>
<thead>
<tr>
<th>Data</th>
<th>Size, bytes</th>
<th>Compression ratio</th>
<th>RMS Elev Error, % of Elev Range</th>
<th>RMS Slope Error, deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>1880</td>
<td>170:1</td>
<td>2.83</td>
<td>3.53</td>
</tr>
<tr>
<td>hill2</td>
<td>1962</td>
<td>163:1</td>
<td>4.06</td>
<td>8.06</td>
</tr>
<tr>
<td>hill3</td>
<td>1739</td>
<td>184:1</td>
<td>1.66</td>
<td>1.65</td>
</tr>
<tr>
<td>mtn1</td>
<td>1979</td>
<td>162:1</td>
<td>3.77</td>
<td>14.0</td>
</tr>
<tr>
<td>mtn2</td>
<td>2006</td>
<td>160:1</td>
<td>4.31</td>
<td>14.1</td>
</tr>
<tr>
<td>mtn3</td>
<td>2004</td>
<td>160:1</td>
<td>4.58</td>
<td>13.3</td>
</tr>
</tbody>
</table>
2.6. Path Planning

We can compute optimal paths using an extension of an A* algorithm that avoid viewsheds while minimizing length and uphill travel. The following montage shows dozens of paths on our six test datasets.

We have a video demonstrating this.

How does our path planning compare to how another contractor's work?

1. We are able to route much more complicated and longer paths (sometimes thousands of pixels long). The following image shows a path computed to avoid hundreds of observers' viewsheds on a 3595x3595 cell.
1. Our metric is more sophisticated.

2. We handle complicated topologies. That is, the hardest part is determining on which side of each observer forbidden zone (viewshed) to put the path. It’s not a simple matter of locally optimizing a geodesic.

3. Tasks to be completed

3.1. Slope

Following the major redirection four months ago, we have been working hard. We have laid out a strategy and have some preliminary results, but have a long way to go. This is an exciting topic that no one else appears to be working on, perhaps because of its difficulty. We are glad to have the opportunity.

One proposed method is shown in this flowchart:
Another is Casual Template Modeling with Slope Preservation:

Lossy terrain compression disregards terrain slope, one of the first order derivatives of the terrain elevation datasets. Our objective is to compress terrain while preserving slope within a user specified tolerance. Our approach is an interplay of several ideas. The way of achieving slope preservation is a smart application of the Bresenham's line algorithm. It is a smart elevation error tolerance scheme, which prevents slope from diverging beyond the tolerance level. The compression method uses a causal template modeling that has excellent decorrelation properties. The parameters for the model however are optimized in a non-traditional way, using an overdetermined system of equation that minimizes the error vector of the model. Another idea is to use asymmetric compression/decompression process. We trade off computational complexity of the compression process for better compression ratios. Longer search for better compression parameters does not affect decompression process, which is swift and easy. Metin Inanc is the primary researcher here.

3.2. Terrain representation - drainage analysis

This current subproject is proceeding as follows.

1. Identify sinks & watershed boundaries with our very fast connected components program.
2. Solve for water flow as a sparse system of linear equations.
3. Invert elevations and solve for ridge rivers.
4. Merge watersheds not blocked by significant ridges.
5. Input ridge and stream points to ODETLAP.

We are using our fast connected-component program for the difficult problem of detecting sinks.

3.3. Code Cleanup

Our efforts have been directed to creating software to validate our hypotheses and obtain results. The code is student-quality. In response to our instructions given at various times, we have not spent the resources to clean it up and package it; instead spending our efforts on developing new ideas.
4. Deliverables

4.1. TIN Code
We have a demo program implementing the accomplished TIN task described above.

4.2. ODETLAP Code
We have prototype code implementing our ODETLAP terrain representation technique described above.

4.3. Siting Code
We have research-quality code to site hundreds of observers on terrain so as to maximize their joint viewshed. Both the observers and targets are a user-defined distance above the terrain. The observers can see out to a use-defined radius of interest. We can do the siting either with or without enforcing that the observers be intervisible to each other. The following figure summarizes this.

![Diagram](image)

Parameters:
- Observer height
- Target height
- Radius of interest
- Intervisibility?

Siting program

Observer positions

Joint viewshed

For one observer, we can compute an error bar on the viewshed as the rounding rules are changed for computing whether or not the line of sight clears the terrain or not. The following image shows this.
The viewsheds that we compute are more detailed than those produced by some of our competitors. Here are some examples.

### 4.4. Path planning to minimize material moved

This algorithm and program are designed to support automated road design. For each representation of the terrain, a road is constructed to minimize the amount of material added/removed, with the constraint that the slope never exceeds a given maximum. That was chosen to be the slope of a straight line connecting the two corners, which is actually quite strict. The A* algorithm is used. The volume of material added/removed in the alternate representation is compared against the volume
added/removed when that same path is applied to the original representation. The percent difference between the two volumes is our error metric. In the table below, the alternate representation for each dataset was ODETLAP with 3000 points. Each dataset required about 6200 bytes, compared to the 320KB uncompressed binary representation. Our representation is lossy; note that that does not impede the path planning.

### Accuracy of path planning on ODETLAP

<table>
<thead>
<tr>
<th>Dataset</th>
<th>% Difference in Volume Added or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1</td>
<td>0.084%</td>
</tr>
<tr>
<td>Hill2</td>
<td>1.536%</td>
</tr>
<tr>
<td>Hill3</td>
<td>0.093%</td>
</tr>
<tr>
<td>Mtn1</td>
<td>2.054%</td>
</tr>
<tr>
<td>Mtn2</td>
<td>0.004%</td>
</tr>
<tr>
<td>Mtn3</td>
<td>0.034%</td>
</tr>
</tbody>
</table>

The average material added or removed was 40 meters or more. Allowing greater slopes, or bridges and tunnels, would reduce this.

The following figures show four paths planned by our program on the datasets Hill2, Mtn1, Mtn2, and Mtn3. In each case, the path was planned from the top left to the bottom corner of the dataset.
Note how complex our paths are, and how they gradually climb up the terrain. Because they are usually not at the level of the terrain, because material is being added or removed during the optimization, this is again a more complex process than merely finding a locally optimal geodesic in a continuous scalar field.

### 4.5. Papers


7. An Improved LLL Algorithm, Franklin T Luk and Daniel M Tracy, *Linear Algebra and its*


15. Preserving the hydrology on simplified terrain. Jonathan Muckell, Marcus Andrade, W. Randolph Franklin, Barbara Cutler, Metin Inanc, and Zhongyi Xie. *(in preparation, Jan 2008)*.


---

### 5. Corrections

*(http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/Ge...)*
A column header in the table in the section *Terrain representation - lossy compression by a factor of 100* was corrected on 2/29/08. *Percent* was originally erroneously reported as *meters*.

Material created by W. Randolph Franklin and others is Copyright © 1994-2007, by the authors, This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 2.5 License · This wiki is pmwiki-2.2.0-beta65 · This page is [http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/GeoStar/Feb2008](http://www.ecse.rpi.edu/Homepages/wrf/pmwiki/GeoStar/Feb2008) · Last modified on March 07, 2008, at 11:05 PM · Now is March 18, 2008, 23:00:07
GeoStar at RPI

DARPA Site Visit

Dr W. Randolph Franklin
Dr Barbara M Cutler

Rensselaer Polytechnic Institute
Troy, NY, 12180

June 18, 2008

1 (518) 276-6077, frankwr@rpi.edu, http://wrfranklin.org/
2 (518) 276-3274, cutler@cs.rpi.edu, http://www.cs.rpi.edu/~cutler/

Outline I

Summary
Crew
Task History
  Tasks in proposal
  Tasks chosen by NGA
  Post-award modifications
  Our response
Tasks accomplished
Tasks Remaining
Unique strengths of RPI
Value to DARPA
Rest of today
Future Ideas
Summary

- Geologically correct terrain data structures and radar siting, HM1582-05-2-0002
- RPI is one of six performers initially under DARPA/DSO/Geo*, PM: Dr Carey Schwartz
- Now in DARPA/IPTO/Geo*, PM: Dr Todd Hughes
- Contract managed by NGA, Dr Ed Bosch

Crew

Current
- Dr W. Randolph Franklin – general secretary
- Dr Barb Cutler – computer graphics
- Zhongyi Xie – ODETLAP, graduated 5/2008
- Dan Tracy – path planning
- Jon Muckell – hydrology, grad 5/2008
- Chris Stuetzle – ODETLAP, hydrology
- Jake Stookey – IBM Blue Gene/L
- 2 new grad students starting in Sept

Past
- Dr Frank Luk – numerical computation (∴ HK 5/2007)
- Dr Caroline Westort – left in 1/2007
- Metin Inanc – segmentation, graduated 5/2008
- Joe Roubal, ESRI – Siting toolkit – contract ended
Tasks in proposal

- Terrain representation
  - Morphological terrain sculpting
  - Overdetermined Laplacian PDE (ODETLAP)
  - Triangulated irregular network (TIN)
  - Lossy compression
- Terrain operators
  - Siting/intervisibility toolkit
  - Trajectory planning
  - Drainage analysis

Tasks chosen by NGA

- In phase I, compress terrain by a factor of 10 with reasonable error.
- In phase II, compress terrain by a factor of 100
- Done; presented earlier.
Post-award modifications

- Make path planning an important theme. (4/2006)
- Don’t productize so much.
- Concentrate on our best terrain representation.
- Do not award remaining budgeted money to ESRI.
- Stop concentrating on extreme terrain compression. Rather, concentrate on representing slope accurately. (11/7/2007)

Our response

- De-emphasized morphological and scooping terrain representation, worked on by Dr Caroline Westort, (*LT great potential, but ST slow*)
- Concentrated on ODETLAP, (*greatest ST potential, current base of most of our work*)
Tasks accomplished

- Morphological terrain sculpting transferred resources to more ST projects
- overdetemined Laplacian PDE (ODETLAP) representation
  Major success, base of most of our current work. Key differentiating factors:
  - smooth,
  - possible progressive transmission,
  - conflate inconsistent partially overlapping data sets,
  - interpolate partial sets of elevation posts,
  - infer local maxima inside the topmost contour.
- Triangulated Irregular Network representation
  - \(10^4 \times 10^4\) points on laptop.
  - finds points in order of importance (unlike competitors)
  - in core (unlike competitors)
- Siting/intervisibility toolkit ESRI produced ArcGIS DLL; development stopped.

Task History

Tasks accomplished – 2

- Lossy compression by a factor of 100. Done.
- Path planning aka Motion planning
  - Done on Level-II DEMs
  - Working on urban LIDAR
  - Better than competitors
    - not just around a small number of blocks, but around a large number of irregular viewsheds while simultaneously minimizing energy
    - not just minimizing energy (geodesic) but also avoiding viewsheds
    - works on hi-res matrices, at least \(3600 \times 3600\)
    - not just quasi-straight paths, but complicated curved paths
Tasks Remaining

- Slope representation. *accurate elevation is insufficient*
- LIDAR compression. *Urban ≠ rural.*
- Hydrology
- Major management challenge
  - Utilize the foundation we’ve laid to produce results.
  - Keep large team together and productive then suddenly stopping next fall.
  - 3 students graduated this spring
  - 2 masters students wisely decided to get jobs while opportunity is there
  - How to start new students so that they can transition to other projects?

Unique strengths of RPI

We have broad competence

- in terrain compression,
- in visibility,
- in hydrology
Value to DARPA

- We make terrain data, and siting and path planning, more available.
- Knowing where to site our observers helps us to shape the battlespace in our favor.
- More compact data can be distributed to smaller computers in the field.
- That helps our people to understand the world and reduces the fog of war.
- This is an asymmetric advantage for our side.

Rest of today

- Students present specific work
  - Dan Tracy, Path planning and slope representation of a compressed terrain
  - Zhongyi & Jake Stookey, Parallel ODETLAP for terrain representation and reconstruction
  - (maybe) Jon Muckell & Chris Stuetzle, Hydrology-aware triangulation of terrain data
- Dr Wolf W. von Maltzahn, Acting Vice of Research
- Dr Cutler and I will present ideas for future
Future Ideas

- conflate global lores with local hires elevation
- play red-blue games with multi-observer siting and path planning, detect and block choke points
- new apps for siting: radio transmitters, micro cells, exit lights, surveillance cameras
- urban multi-observer siting, in 3D
- conflation, compression and data fill-in of urban geometric data while preserving structure and the laws of formation
  - roads, rivers are continuous and usually don’t dead-end
  - size depends on catchment
- modify the real world to enhance the goal (visibility, motion, ...)

RPI Geo* Final Report
Geo* at RPI – Jun 2008
Hydrology-Aware Triangulation of Terrain Data

Jonathan Muckell, W. Randolph Franklin, Marcus Andrade, Barbara Cutler, Melin Inanc, Zongyi Xie, Daniel M. Tracy

*Department of Electrical and Computer Engineering, Rensselaer Polytechnic Institute
*Department of Computer Science, Rensselaer Polytechnic Institute

**Problem**
- **Large Dataset Size**: Terrain data is being sampled at over increasing resolutions over larger geographic areas requiring special compression techniques to manipulate the data.
- **Sampling Issues**: Dataset inaccuracies due to insignificant elevation sampling and data collection errors impede water flow by causing small and unrealistic watersheds.
- **Measurement Uncertainty**: Typically, terrain compression algorithms like the progressive DIMS (mean square error) and maximum error. These metrics fail to capture whether a reconstructed terrain preserves the drainage network.

**Measuring Hydrology Error**

- **Total Downward Energy**
  \[ E_{\text{down}} = \sum (E_i - E_{i+1}) \]
- **Total Upward Energy**
  \[ E_{\text{up}} = \sum (E_{i+1} - E_i) \]
- **Error**
  \[ \text{Error} = \frac{E_{\text{up}}}{E_{\text{down}}} \]

To compute the accuracy of the drainage network, the gradient, amount of flow contributing cells, and whether the flow is traveling uphill or downhill are taken into account. The total downward energy and upward energy is computed as a summation of the gradient (E - E_i), where E_i is the original elevation matrix and E is the receiving elevation matrix where each cell contains the elevation of the adjacent cell in E_i that is receiving the water flow. The gradient is weighted by the amount of flow (variable W). The final Error is determined as the ratio of the total upward energy divided by the total downward energy.

**Method**
- **Ridge River Network**: Inverting the terrain and analyzing the drainage network provides an approximation.
- **Denoising**: Reduces the number of points required to represent each ridge-river segment. The refined points can be stored and further compressed to which a given error tolerance.
- **Eliminating Degeneracies**: Creates a triangle mesh of triangular facets from overlapping planes that maximizes the minimum angle of all triangles in the image.
- **Draping Network Program**: Determines the number of cells that contribute flow to every cell in the terrain using a system of linear equations.

**Results**
- **Realistic Waterways and Drainage Networks**
- Better compression then using other GIS data structures such as Triangulated Irregular Networks (TIN)
- Developed a quantitative measurement for measuring the amount of hydrology error using a potential energy metric
- Efficient drainage network computation based on a system of linear equations. The resulting drainage often contains longer and more realistic drainage networks than ArcGIS which is typically regarded as the industry standard.
- Simple, fast and effective computation of the ridge network. Having an accurate representation of the ridge network can assist compression algorithms and also has applications in siting, path planning, and hydrology.

[Image of a map showing hydrology error and triangulation process]

HydroTIA after nine iterations. Notice ridges and valleys, essential for hydrology computation are preserved along triangle edges.

[Image of a map with hydrology network before and after triangulation]
Hydrology-Aware Triangulation of Terrain Data

Jonathan Muckell

Outline

1. How we define hydrology features
2. How we compress the terrain to preserve these features
3. How we measure the amount of hydrology error
Terrain Representation Problems

- **Large Dataset Sizes:** Terrain data is being sampled at ever increasing resolutions over larger geographic areas requiring special compression techniques to manipulate the data.

- **Sampling Issues:** Dataset inaccuracies due to insignificant resolution sampling and data collection errors impedes water flow causing small and unrealistic watersheds.

- **Measuring Effectiveness:** Typically, terrain compression algorithms seek to minimize RMS (root mean square) and maximum error. These metrics fail to capture whether a reconstructed terrain preserves the drainage network.

---

Triangulated Irregular Networks (TIN)

- Create a triangular mesh that captures terrain topology
- TIN minimizes maximum error
  - Selects points one at a time based on maximum distance from reconstruction
- Lots of points are placed in regions that have the most structure
What is Hydrology-Aware?

A targeted compression geared towards preserving the hydrology features of a specified terrain

The Ridge-River Network

Step 1: Using the original elevation matrix, set flow directions based on steepest descent flow.

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
1 & 7 & 1 & 1 \\
3 & 9 & 5 & 1 \\
1 & 1 & 1 & 1 \\
6 & 7 & 1 & 1 \\
\end{bmatrix}
\]

Step 2: Using a $N^2$ by $N^2$ sparse matrix (where $N$ is the size of the $N$ by $N$ DEM) set non-zero elements equal to the identity and adjacent neighbors the contribute flow.

\[
\begin{bmatrix}
X_1 & X_2 & X_3 & X_4 \\
X_5 & X_6 & X_7 & X_8 \\
X_9 & X_{10} & X_{11} & X_{12} \\
X_{13} & X_{14} & X_{15} & X_{16} \\
X_{17} & X_{18} & X_{19} & X_{20} \\
\end{bmatrix}
\]

Bounded by $2N^2$ non-zero entries
The Ridge-River Network

\[
\begin{pmatrix}
X_1 & X_2 & X_3 \\
X_4 & X_5 & X_6 \\
X_7 & X_8 & X_9
\end{pmatrix} = \begin{pmatrix} 6 & 3 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 3 \end{pmatrix}
\]

**Step 3:** Solve the system. Each cell now contains an integer corresponding to the number of cells the contribute flow to that cell.

Time to Initialize and Solve the Linear Systems

Accounting for Flat Areas

- Detect using a very fast connected components program
- Breadth-first search to assign directions
Accounting for Flat Areas

Locate the spill points – Perform a Breadth-First search out from each spill point. This divides the plateau into separate regions where water escapes.

Assign Flow Directions

Alter the terrain DEM. Recompute part of the flow routing program to take into account the new data.
Our Method vs ArcGIS

Ridge Network is found by inverting the terrain and running the drainage network program

\[ I_e = \frac{\text{Max}(E) - E + \text{Min}(E)}{\text{Max}(E) - \text{Min}(E)} \]

Equation for inverting the terrain
Douglas-Peucker Line Refinement

- Removes insignificant points. Point can deviate at most tolerance distance away from the original.

Triangulate the Terrain

- Tolerance
- Douglas-Peucker
- Ridge-River Network
- Delaunay Triangulation
- Error above threshold?
- Add points with highest error
- Flow Direction and Accumulation
- Drainage Network Program
- Original Terrain
Create network of non-overlapping, irregular sized and oriented triangles while avoiding small "silvers".

Problem: We don't want all three vertices to attach to all river points. We need to have at least one high point in every triangle.

Solution: Triangulate the terrain and find the these triangles, then add the maximum elevation point that occurs in these triangles.
Using a Different Reconstruction these areas are beneficial

Merge watersheds that are not blocked by a significant ridges

Compute Drainage Network on the Reconstructed Terrain
Measuring Hydrology Error

To determine the amount of hydrology error lost during terrain simplification the drainage network computed on the reconstructed geometry is mapped onto the original elevation matrix.
Measuring Hydrology Error

\[ \text{Energy}_{\text{Down}} = \sum (E_i - E_{i-1}) \times W_i \]
\[ \text{Energy}_{\text{Up}} = \sum (E_i - E_{i+1}) \times W_i \]
\[ \text{Error} = \frac{\text{Energy}_{\text{Up}}}{\text{Energy}_{\text{Down}}} \]

- Error is reduced for flow traveling downhill, increased for flow traveling uphill.
- Error is determined by gradient and amount of flow.

- k number of points with the highest error are added at each iteration
- Points that are close together are not added.

RESULTS

![Graphs showing energy balance and error analysis](image)

RPI Geo* Final Report 682/919 Jun 2008 Muckell Talk
References


References


Path Planning and Slope Representation of a Compressed Terrain
Dan Tracy
Rensselaer Polytechnic Institute

Outline

- Terrain Compression
  - ODETLAP
  - ODETLAP with slope equations
- Smugglers and Border Guards
  - Multiple Observer Siting
  - Path Planning
Motivation

- Terrain representation
- Smugglers and border guards

Terrain Compression

- Must evaluate the information loss of the compression
- Reconstitute the terrain from the compressed data to obtain the alternate representation
- Compare the alternate representation against the original
- Simple metrics such as RMS and max elevation error
- More complex metrics such as visibility and path planning
Outline

- Terrain Compression
  - ODETLAP
  - ODETLAP with slope equations
- Smugglers and Border Guards
  - Multiple Observer Siting
  - Path Planning

ODETLAP

- Given an n-by-n grid of points, and the elevations of only k of the points are given
- Goal: interpolate the unknown points
- Solution: ODETLAP
  - An overdetermined system of linear equations that approximates the Laplacian PDE
ODETLAP procedure

- Set each point equal to the average of its four neighbors.
- Set each known point equal to its known value. These points are allowed to be changed.
- Smaller R: More accurate solution
- Larger R: Smoother solution
- n²+k equations for n² unknowns
Benefits of ODETLAP

- Can fill in gaps in the data
- Can infer local maxima
- Can conflate inconsistent data
- Input points become invisible in the surface

ODETLAP compression

- Select a set of control points → compressed terrain
  - Use ODETLAP to help select the points.
- Reconstruct the terrain by applying ODETLAP to the control points.
Slopes

- Zevenbergen–Thorne method
  - Vector of the difference between the northern neighbor and southern neighbor.
  - Vector of the difference between the western neighbor and eastern neighbor
  - Cross product of the vectors: slope normal
- Slope error
  - Angular difference between the slope normals

Test Data
(400x400 DTED II)

W111
N31 subsets

<table>
<thead>
<tr>
<th>Hill1</th>
<th>Hill2</th>
<th>Hill3</th>
</tr>
</thead>
</table>

W121
N38 subsets

<table>
<thead>
<tr>
<th>Mtn1</th>
<th>Mtn2</th>
<th>Mtn3</th>
</tr>
</thead>
</table>
Algorithm Outline

- Original DEM data

- Use TIN to find initial point set S

- Compare two elevation matrices

- Run ODETLAP to get an approximation

- If not good enough

- Find the points with the largest slope errors and add them into point set S.

ODETLAP Results

<table>
<thead>
<tr>
<th>Data</th>
<th>Mean SE</th>
<th>Max SE</th>
<th>RMS SE</th>
<th>Below 2º Perc.</th>
<th>Total Pts</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill2</td>
<td>2.58</td>
<td>16.78</td>
<td>3.00</td>
<td>41%</td>
<td>16960</td>
<td>25195</td>
</tr>
<tr>
<td>mtn1</td>
<td>2.62</td>
<td>17.87</td>
<td>3.00</td>
<td>57%</td>
<td>36280</td>
<td>49035</td>
</tr>
<tr>
<td>mtn2</td>
<td>2.62</td>
<td>14.87</td>
<td>3.00</td>
<td>57%</td>
<td>38400</td>
<td>47428</td>
</tr>
<tr>
<td>mtn3</td>
<td>2.62</td>
<td>24.57</td>
<td>3.00</td>
<td>38%</td>
<td>33880</td>
<td>46918</td>
</tr>
</tbody>
</table>
Outline

- Terrain Compression
  - ODETLAP
  - ODETLAP with slope equations
- Smugglers and Border Guards
  - Multiple Observer Siting
  - Path Planning

Slope equations

- Original ODETLAP equations:
  \[ z_{i,j} = \frac{z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1}}{4} \]
  \[ z_{i,j} = h_{i,j} \]
- Slope equations:
  \[ z_{i+1,j} - z_{i-1,j} = h_{i+1,j} - h_{i-1,j} \]
  \[ z_{i,j+1} - z_{i,j-1} = h_{i,j+1} - h_{i,j-1} \]
Slope compression

Point selection algorithm
- Input: elevation matrix A, parameter PtsPerIter
- Generate the ODETLAP equations in the following order:
  - Create the known elevation equations. Apply a regularly spaced grid to the terrain and record the known elevation values.
  - Iteratively:
    - Solve the system formed by the Laplacian approximations, the known elevation equations, and the slope equations.
    - Choose PtsPerIter number of points that have the largest slope error.
    - For each chosen point, add two slope equations to the system.

Results

Number of chosen points out of 160,000 on mtn2 vs. Avg/Max slope error

June 18, 2008
Storage requirements

- Elevation equations: For each point, 3 values are needed: the x,y indices and the elevation.
- Slope equations: For each point, 4 values are needed: the x,y indices of that point, the delta-h in the x-direction, and the delta-h in the y-direction.

Encoding

- Run Length Encoding for the x,y indices
- Delta-h’s:
**k-means clustering**

Median values for k-bits (degrees):
- 1 bit: -25.0, 20.1
- 2 bits: -35.0, -18.4, 11.3, 33.7
- 3 bits: -42.0, -33.7, -25.0, -13.1, 5.7, 20.1, 31.0, 42.0
- 4 bits: -46.8, -40.9, -36.9, -32.3, -28.1, -23.4, -16.7, -7.6, 1.9, 9.5, 18.4, 25.0, 31.0, 35.0, 40.9, 47.7

---

**Encoding**

![Graph showing encoding efficiency for different data compression levels](image-url)

- 1 bit per delta-z
- 2 bits per delta-z
- 3 bits per delta-z
- 4 bits per delta-z
- 5 bits per delta-z
- 6 bits per delta-z

June 18, 2008
Outline

- Terrain Compression
  - ODETLAP
  - ODETLAP with slope equations
- Smugglers and Border Guards
  - Multiple Observer Siting
  - Path Planning

Multiple Observer Siting

Site a group of observers so as to maximize the amount of visible terrain. e.g.: place cell phone towers in order to optimize the coverage area.
Outline

- Terrain Compression
  - ODETLAP
  - ODETLAP with slope equations
- Smugglers and Border Guards
  - Multiple Observer Siting
  - Path Planning

Path Planning

- Smuggler’s Path: Find the shortest path between two given points while trying to avoid detection by the observers.
- A* algorithm
- Add penalty for going uphill.
Range of Motion

A straightforward application of the A* algorithm results in the Chebyshev distance being minimized, rather than the Euclidean distance.

June 18, 2008

Path Planning

- New approach: Two pass system
- First pass: Plan a path that minimizes Chebyshev distance.
- Second pass: Only include points from the first path in the search space.
- Not guaranteed to be optimal, but in practice it often is.

June 18, 2008
Cost Metric

- Cost of moving from one cell to an adjacent cell:
  
  \[ \text{Cost} = \sqrt{(h^2 + v^2)} \times \text{Slope Penalty} \times \text{Visibility Penalty} \]

- \( h \) is the horizontal distance.
- \( v \) is the elevation difference.
- SlopePenalty is \( \frac{1 + \frac{v}{h}}{} \) when going uphill and 1 otherwise.
- VisibilityPenalty is 1 if the new cell is not visible and 100 otherwise.
Error Metrics

1) Viewshed Error: Area of the symmetric difference between the two joint viewsheds

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ODETLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>0.22%</td>
</tr>
<tr>
<td>hill2</td>
<td>1.29%</td>
</tr>
<tr>
<td>hill3</td>
<td>0.09%</td>
</tr>
<tr>
<td>mtn1</td>
<td>7.52%</td>
</tr>
<tr>
<td>mtn2</td>
<td>5.76%</td>
</tr>
<tr>
<td>mtn3</td>
<td>5.18%</td>
</tr>
</tbody>
</table>

June 18, 2008

Error Metrics

2) Path Cost Error: Difference of the costs of the paths computed on the original and alternate representations.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ODETLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>3.23%</td>
</tr>
<tr>
<td>hill2</td>
<td>4.21%</td>
</tr>
<tr>
<td>hill3</td>
<td>8.89%</td>
</tr>
<tr>
<td>mtn1</td>
<td>7.25%</td>
</tr>
<tr>
<td>mtn2</td>
<td>6.10%</td>
</tr>
<tr>
<td>mtn3</td>
<td>10.36%</td>
</tr>
</tbody>
</table>

Alternate       Original

June 18, 2008
Error Metrics

3) Path Visibility Error: If the path computed on the alternate representation is transferred to the original, how much is the visibility preserved?

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>0.00%</td>
</tr>
<tr>
<td>hill2</td>
<td>0.00%</td>
</tr>
<tr>
<td>hill3</td>
<td>0.00%</td>
</tr>
<tr>
<td>mtn1</td>
<td>0.52%</td>
</tr>
<tr>
<td>mtn2</td>
<td>1.79%</td>
</tr>
<tr>
<td>mtn3</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

June 18, 2008

---

Hill 3

Elevation range: 500 m
Elevation stddev: 59 m

Original June 18, 2008
Alternate
Mtn 1
Elevation range: 1040 m
Elevation stdev: 146 m

Original       Alternate
June 18, 2008

Mtn 2
Elevation range: 953 m
Elevation stdev: 152 m

Original       Alternate
June 18, 2008
Ottawa LIDAR Data

- Used ODETLAP to convert the point cloud to a 2000x2000 raster.

Ongoing Research

- Compute multiple paths by specifying various pairs of start/end points.
- Make sure that the hidden areas are disconnected.
- Moving observers: Compute paths for tourists, smugglers.
- Red/Blue games: The blue team tries to hide; the red team tries to find them.
Automated Road Construction

- Connect two points by constructing a road between them
- The slope of the road cannot exceed a given maximum.
- The goal is to place the road so as to minimize the amount of material removed/added.

June 18, 2008

Automated Road Construction as an Error Metric

Results:
Compression scheme was ODETLAP with 3000 control points.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>% Difference in Volume Added or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill1</td>
<td>0.884%</td>
</tr>
<tr>
<td>Hill2</td>
<td>1.536%</td>
</tr>
<tr>
<td>Hill3</td>
<td>0.093%</td>
</tr>
<tr>
<td>Mtn1</td>
<td>2.054%</td>
</tr>
<tr>
<td>Mtn2</td>
<td>0.004%</td>
</tr>
<tr>
<td>Mtn3</td>
<td>0.034%</td>
</tr>
</tbody>
</table>

June 18, 2008


**Constructed Roads**

- Hill2
- Mtn1
- Mtn2
- Mtn3

June 18, 2008

---

**Ongoing Research**

- Account for the distance traveled by the bulldozer.
- Restrict the movement of the bulldozer. (e.g.: maximum slope it can traverse)
- The bulldozer can dispose of material at a “sink” or obtain material from a “source,” which are at fixed locations.
- Multiple bulldozers

June 18, 2008
Summary

- We created new application-specific error metrics for terrain compression
- We have computed good smuggler paths, including on urban LIDAR data.
- Our ODETLAP scheme performs well on the metrics and slopes.

June 18, 2008

Thank You

Smuggler’s Video.mpeg

June 18, 2008
Geostar at RPI
Arlington VA Review

Dr W. Randolph Franklin
Dr Barbara M Cutler

Rensselaer Polytechnic Institute
Troy, NY, 12180

August 11, 2008

Summary

- Geologically correct terrain data structures and radar siting, HM1582-05-2-0002
- RPI is one of six performers initially under DARPA/DSO/Geo*, PM: Dr Carey Schwartz
- Now in DARPA/IPTO/Geo*, PM: Dr Todd Hughes
- Contract managed by NGA, Dr Ed Bosch
- All our material is online (passworded) at

  http://wrfranklin.org/pmwiki/GeoStar/GeoStar
Crew

Current
- Dr W. Randolph Franklin – general secretary
- Dr Barb Cutler – computer graphics
- Dan Tracy – path planning
- Chris Stuetzle – ODETLAP, hydrology
- Jake Stookey – IBM Blue Gene/L
- Eddie Yam — starting grad student, 9/2008
- another starting grad student, 9/2008

Past
- Dr Frank Luk – numerical computation (→HK 5/2007)
- Dr Caroline Westort – left in 1/2007
- Dr Metin Inanc – segmentation, graduated 5/2008
- Joe Roubal, ESRI – Siting toolkit – contract ended
- Jon Muckell – hydrology, grad 5/2008
- Zhongyi Xie – ODETLAP, graduated 5/2008

Tasks in proposal

- Terrain representation
  - Morphological terrain sculpting
  - Overdetermined Laplacian PDE (ODETLAP)
  - Triangulated irregular network (TIN)
  - Lossy compression
- Terrain operators
  - Siting/intervisibility toolkit
  - Trajectory planning
  - Drainage analysis
Tasks chosen by NGA

- In phase I, compress terrain by a factor of 10 with reasonable error.
- In phase II, compress terrain by a factor of 100
- Done; presented in Oct 2007.
- Uncompressed binary file size: 320KB
- Uncompressed ascii file size: 800KB–2400KB

Post-award modifications

- Make path planning an important theme. (4/2006)
- Don’t productize so much.
- Concentrate on our best terrain representation.
- Do not award remaining budgeted money to ESRI.
- Stop concentrating on extreme terrain compression. Rather, concentrate on representing slope accurately. (11/7/2007)
Our response

- De-emphasized morphological and scooping terrain representation, worked on by Dr Caroline Westort, *(long term great potential, but short term slow)*
- Concentrated on ODETLAP, *(greatest short term potential, current base of most of our work)*

Tasks accomplished

- **Morphological terrain sculpting** transferred resources to more ST projects
- **overdetermined Laplacian PDE (ODETLAP) representation**
  - Major success, base of most of our current work. Key differentiating factors:
  - smooth,
  - possible progressive transmission,
  - conflate inconsistent partially overlapping data sets,
  - interpolate partial sets of elevation posts,
  - infer local maxima inside the topmost contour.

- **Triangulated Irregular Network representation**
  - $10^4 \times 10^4$ points on laptop.
  - finds points in order of importance (unlike competitors)
  - in core (unlike competitors)

- **Siting/intervisibility toolkit** ESRI produced ArcGIS DLL; development stopped.
Tasks accomplished – 2

- Lossy compression by a factor of 100. Done.
- Path planning aka Motion planning
  - Done on Level-II DEMs
  - Working on urban LIDAR
  - We believe to be better than competitors
    - not just around a small number of blocks, but around a large number of irregular viewsheds while simultaneously minimizing realistic non-symmetric energy.
    - works on hi-res matrices, processing 3600 × 3600 cells in 2006.
    - not just quasi-straight paths, but complicated curved paths

Tasks Remaining

- Slope representation. *accurate elevation is insufficient*

- LIDAR compression. *Urban ≠ rural.*
Value to DARPA

- We make terrain data, and siting and path planning, more available.
- Knowing where to site our observers helps us to shape the battlespace in our favor.
- More compact data can be distributed to smaller computers in the field.
- That helps our people to understand the world and reduces the fog of war.
- This is an asymmetric advantage for our side.

Slope Accuracy on Compressed Terrain — Why consider slope?

Slope is important for
- mobility
- erosion
- aircraft
- visibility
- recognition
Bad commercial slope representation

Commercial SW:

Photo:
Accurate elevations \(\not\Rightarrow\) accurate slopes

- Ignoring errors, slope is simply \(f'(x)\)
- But \(\lim sup_{i \to \infty} |(f_i(x) - f(x))| \to 0\), gives no guarantees about \(\lim sup_{i \to \infty} |(f'_i(x) - f'(x))|\)
- Consider two approximations to \(y(x) = 0\)
- Elevation got better but slope got worse.

ODETLAP – Overdetermined Laplacian Method

Fundamental representation for this work
- Small set of posts \(\Rightarrow\) complete matrix of posts
- Overdetermined linear system:
  - \(z_{ij} = h_{ij}\) for known points,
  - \(4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1}\) for all nonborder points.
  - Emphasize accuracy or smoothness by weighting the two types of equations differently.
- Fills contours to a grid.
- Fill missing data holes.
- Handles
  - incomplete contours,
  - complete contours,
  - kidney-bean contours,
  - isolated points,
  - inconsistent data.
ODETLAP hard example

- input: contours with sharp corners
- output: smooth silhouette edges, inferred top

ODETLAP process

Input

- 400x400 matrix of elevations
- contour lines
- any user-supplied points, even inconsistent

ODETLAP point selection

- Small point set \(~1000\)
- Compressed distributed data
- Reconstructed data

ODETLAP terrain reconstruction

- 400x400 matrix of elevations
ODETLAP TIN+greedy point insertion

- Use incremental TIN to find initial set $\mathcal{P}$ of approx 1000 important points.
- Fit surface with ODETLAP.
- If it's good enough, then stop.
- Find approx 30 worst points, and insert into $\mathcal{P}$.
- Loop back to step 2.

*Why 30?* Efficiency; ODETLAP takes minutes per run.  
*Forbidden zone concept:* In the same step, don’t insert very close points.

Coding the points

Goal is min size not fewest points
- *Coding* $\{(x, y, z)\}$ to minimize size is as important as selecting the points.
- Various approaches were presented elsewhere.
- Using more points is good, if they can be coded better.
- E.g., regular grid of points.
- If progressive transmission is not desired, then, for irregular points, use compressed bitmap for $\{(x, y)\}$ and *bzip2* for $(z)$. 
Slope definition, accuracy

- Zevenbergen-Thorne \( \left( (p_{i-1,j} - p_{i+1,j}) \times (p_{i,j-1} - p_{i,j+1}) \right)_z \)
- \( p_{ij} \) not used

Limits of slope accuracy

- 1m elevation resolution
- 30m post spacing
- slope precision: \( \arctan \left( \frac{1}{30} \right) \approx 3\% \approx 2^\circ \)

Info content

- Slope’s autocorrelation distance is smaller than elevation’s
- However, slope has less relative precision.

Level-II sample datasets

400 × 400 elevation matrices, elevation range

Hill1 505m Hill2 745m Hill3 500m
Mtn1 1040m Mtn2 953m Mtn3 788m
Idea 1: Pin down the elevation at sets of close points

- When inserting a point into known set, also insert some adjacent points
- *Thesis:* that will force the slope to be accurate there.
- *Not really.*
- *Analogy* Lagrangian interpolation.

Keep trying.

---

Idea 2: Extend ODETLAP

- Explicitly incorporate slope
- New overdetermined linear system:
  - unknowns: $z_{ij}$
  - known:
    - some $h_{ij}$,
    - some $\Delta x h_{ij} \triangleq h_{i-1,j} - h_{i+1,j}$,
    - some $\Delta y h_{ij} \triangleq h_{i,j-1} - h_{i,j+1}$,
  - for all nonborder points:
    $$4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1}$$
  - for known $h_{ij}$: $z_{ij} = h_{ij}$
  - for known $\Delta x h_{ij}$ and $\Delta y h_{ij}$:
    $$z_{i-1,j} - z_{i+1,j} = \Delta x h_{ij}$$
    $$z_{i,j-1} - z_{i,j+1} = \Delta y h_{ij}$$
Mtn2 experiments

Slope error vs number of points

Mtn2 experiments
Number of sufficiently accurate points (out of 160K points)
Mtn2 experiments

Slope error vs compressed file size

- mtn2, ODETLAP with slope equations, lossy encoding of delta-z

Slope compression conclusions

- We’ve done well on this task assigned in Oct 2007.
- Terrain can be compressed to as to represent slope more accurately.
- ODETLAP represents terrain efficiently.
- Faster ODETLAP; now using RPI’s IBM Blue Gene/L (32K processors, #7 in June 2007 top500.org list).
Path planning

• Carey Schwartz asked for this in 4/2006.
• Fun to combine various projects we’ve worked on.
• multiobserver siting + path planning + surface compression with ODETLAP
• unique feature of our path planning: plans around complicated obstacles (viewsheds) while minimizing complex non-symmetric objective:

\[ C = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \cdot \left( 1 + \max \left( 0, \frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}} \right) \right) \cdot (1+100v) \]
Path planning on Ottawa LIDAR data

Video of elevation and slope errors during point insertion

See how the elevation error and slope error optimize as ODETLAP points are greedily inserted.
Parallel ODETLAP

- using RPI's IBM Blue Gene/L (32K processors, #7 in June 2007 top500.org list).
- Process overlapping patches of terrain, one patch per processor.
- Blend the resulting arrays of computed elevations into one large array.

Challenges:
- Each processor has limited memory and no VM.
- Manage the communication to avoid bottlenecks.

Preliminary results are on a 128-processor AMD Opteron.

13 · 10^6 points ⇒ 10^4 × 10^4 raster

Our parallel version of ODETLAP was used to generate a dense 10000 × 10000 raster (B) from raw urban LIDAR data consisting of 13 million points (A) quickly on a cluster of 128 2.6 GHz AMD Opteron processors.
Overlapping patches

We apply ODETLAP to overlapping layers of patches. In this example, 4 patches would be enough to cover the entire heightmap, but we would see errors at the patch edges. Instead, we run ODETLAP on 9 overlapping patches.

Merging the patches

Our final step is to merge the overlapping patches (A) into the complete reconstructed elevation map (B).
Benefits of patches

Execution time for the non-patch version of ODETLAP grows quadratically with the number of pixels, while the patch version of ODETLAP has a linear growth. This improvement is gained while running serially.

Parallel speedup

Reconstruct a 2000 × 2000 grid from 1% of the original points by running ODETLAP on an 2.6GHz AMD Opteron cluster. Observe the linear decrease in running time for up to 128 processors. After that the file I/O overhead dominates. The algorithm includes a central process to merge results, but this process is not included in the results.
Future Ideas

- conflate global lo-res with local hi-res elevation
- play red-blue games with multi-observer siting and path planning, detect and block choke points
- new apps for siting: radio transmitters, micro cells, exit lights, surveillance cameras
- urban multi-observer siting, in 3D
- conflation, compression and data fill-in of urban geometric data while preserving structure and the laws of formation
  - roads, rivers are continuous and usually don’t dead-end
  - size depends on catchment
- Long term goal: procedural terrain representation, where the math captures the structure.
- modify the real world to enhance the goal (visibility, motion, ...)

Publications

13 publications, workshop presentations, or posters so far. Several more in review, and being prepared.
Commercialization

Two 2007.2 SBIRs clearly based on this research.

- **A07-123 Novel Representations of Elevation Data**
  Two phase I awards:
  - W9132V-08-C-0012 to Andrews Space, Inc.
  - W9132V-08-C-0013 to Numerica Corp.

- **A07-126 Optimal Intervisibility Site Selection**; cited me four times. Phase I award W9132V-08-C-0005 to Toyon Research Corp.

(Unfortunately) I have no connection to any of those companies.

The fact that someone in the ARO considers this work important enough to issue solicitations to extend it says that DARPA is succeeding in having Geo* research transition to the Army.

Unique strengths of RPI

We have broad competence

- in terrain compression,
- in visibility,
- in parallel computation
Followon Projects

- Our reputation led to our being invited to host the 18th Fall Workshop in Computational Geometry here at RPI on Fri 10/31 and Sat 11/1/2008. We are very hopeful of $12,034 NSF support. I would like to supplement that with some of my Geo* money that would have been spent on grad students who left early.

- We are very hopeful\(^3\) of being funded on an NSF Cyber-enabled Discovery and Innovation (CDI) Fundamental Terrain Representations and Operations. That program has an expected 2% (1 in 50) funding rate this year.

- We are eager to apply our demonstrated expertise to continue to work with DARPA and NGA.

\(^3\)revised budget and abstract requested
5 Doctoral Thesis

Compressing terrain elevation datasets – Metin Inanc 729
COMPRESSING TERRAIN ELEVATION DATASETS

By

Metin Inanc

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

Wm. Randolph Franklin, Thesis Adviser

Barbara M. Cutler, Member

George Nagy, Member

Michael Wozny, Member

Rensselaer Polytechnic Institute
Troy, New York

May 2008
(For Graduation May 2008)
COMPRESSING TERRAIN ELEVATION DATASETS

By

Metin Inanc

An Abstract of a Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

The original of the complete thesis is on file in the Rensselaer Polytechnic Institute Library

Examining Committee:

Wm. Randolph Franklin, Thesis Adviser
Barbara M. Cutler, Member
George Nagy, Member
Michael Wozny, Member

Rensselaer Polytechnic Institute
Troy, New York

May 2008
(For Graduation May 2008)
# CONTENTS

LIST OF TABLES ................................................................. iii
LIST OF FIGURES .............................................................. v
ACKNOWLEDGMENT ............................................................. vii
ABSTRACT ................................................................. viii

1. INTRODUCTION ......................................................... 1
   1.1 Motivation .......................................................... 1

2. RELATED WORK .......................................................... 4
   2.1 Terrain Representation ............................................. 4
      2.1.1 Contour Representation ...................................... 4
      2.1.2 DEM Representation .......................................... 5
      2.1.3 TIN ............................................................ 7
         2.1.3.1 Simulation of Simplicity ............................. 10
      2.1.4 Related Techniques .......................................... 11
         2.1.4.1 Level of Detail ...................................... 11
         2.1.4.2 ODETLAP .............................................. 12
         2.1.4.3 Terrain Slope ...................................... 15
   2.2 Context Dilution .................................................... 16
   2.3 The Modern Compression Paradigm .................................. 17
   2.4 Generic Compression Algorithms ................................... 17
      2.4.1 Huffman Coding ........................................... 18
      2.4.2 LZ Compression ........................................... 19
      2.4.3 Arithmetic Coding ........................................ 20
      2.4.4 High-Order Modeling ..................................... 23
   2.5 Image Compression .................................................. 24
      2.5.1 The JPEG Family .......................................... 25
      2.5.2 Other Image Compression Techniques ...................... 28
      2.5.3 Medical Images and High Dynamic Range Images .......... 29
   2.6 Document Compression .............................................. 30
3. THE ODETCOM METHOD ......................................................... 32
  3.1 The ODETCOM Model ..................................................... 32
    3.1.1 The ODETCOM Predictor .......................................... 33
    3.1.2 Computing the Predictor Coefficients ......................... 34
    3.1.3 Toy Example ...................................................... 37
    3.1.4 Coding the Residual ............................................. 42
      3.1.4.1 The Unary Code ........................................... 42
      3.1.4.2 The Indexed (Rice) Code ................................ 42
      3.1.4.3 Coding Negative Numbers ................................ 44
      3.1.4.4 Near-Lossless Coding .................................... 44
  4. RESULTS ................................................................. 47
    4.1 ODETCOM on 30 m Horizontal Resolution Datasets ............ 47
      4.1.1 Lossless Results ............................................. 47
      4.1.2 Near-Lossless Results ...................................... 48
    4.2 ODETCOM on 10 m Horizontal Resolution Datasets ............ 48
      4.2.1 Lossless Results ............................................. 49
      4.2.2 Near-Lossless Results ...................................... 49
    4.3 ODETCOM on 3 m Horizontal Resolution Datasets ............ 51
      4.3.1 Lossless Results ............................................. 51
      4.3.2 Near-Lossless Results ...................................... 52
    4.4 Extreme Cases .................................................... 52
      4.4.1 Compressing an All-Zero Dataset ........................... 53
      4.4.2 Compressing Noise ........................................... 53
    4.5 Application on Images ............................................. 54
  5. CONCLUSION AND FUTURE WORK ........................................ 56
LITERATURE CITED ............................................................ 56
LIST OF TABLES

2.1 A Huffman code for three symbols A, B and C. ....................... 18

2.2 Lossless JPEG predictors based on the samples a, b and c of the causal template in Figure 2.12 on page 26. .......................... 26

3.1 Indexed (Rice) Coding with Different Offsets, Offset of 0 bits corresponds to Unary Coding ......................... 44

3.2 Bucketing for near-lossless coding with error bound of 1 ........ 45

4.1 The breakdown on the lossless compression performance (30 m datasets). 48

4.2 The aggregate performance of lossless ODETCOM and JPEG on 1000 30 m datasets. ........................... 48

4.3 The breakdown on the near-lossless compression performance (30 m datasets) ........................................ 49

4.4 The aggregate performance of near-lossless ODETCOM on 1000 30 m datasets. ................................. 49

4.5 The breakdown on the lossless compression performance (10 m datasets). 50

4.6 The aggregate performance of lossless ODETCOM and JPEG on 430 10 m datasets. ......................... 50

4.7 The breakdown on the near-lossless compression performance (10 m datasets) ........................................ 50

4.8 The aggregate performance of near-lossless ODETCOM on 430 10 m datasets. ................................. 50

4.9 The breakdown on the lossless compression performance (3 m datasets). 51

4.10 The aggregate performance of lossless ODETCOM and JPEG on 72 3 m datasets. ............................ 51

4.11 The breakdown on the near-lossless compression performance (3 m datasets) ........................................ 52

4.12 The aggregate performance of near-lossless ODETCOM on 72 3 m datasets. ................................. 52

4.13 The performance of lossless ODETCOM and JPEG on the all-zero dataset. ........................................ 53
4.14 The performance of lossless ODETCOM and JPEG on the uniform noise dataset. ................................................................. 53
4.15 The performance of near-lossless ODETCOM on the uniform noise dataset. ................................................................. 54
4.16 Bitrate and compression ratio for the cameraman image ODETCOM vs. JPEG variants. ....................................................... 55
LIST OF FIGURES

1.1 From the Smithsonian Traveling Exhibition “Earth to Space”, the RPI Library, January 2008. .......................... 2

2.1 Hypsography of the Earth. ........................................... 5

2.2 Excerpt of the topographic map depicting Crane Mountain near Thurman, NY (Adirondacks). Elevations are in feet. ............... 6

2.3 Kauia Touristic Map and Kauai DEM (rendered with MICRODEM [22]). 8

2.4 A close-up of a TIN overlayed on a terrain rendering. .................. 10

2.5 Filling a circular hole (R=100) of missing data using ODETLAP. Note the local maxima inferred inside the circle. .................... 13

2.6 Filling a circular hole (R=40) of missing data using ODETLAP with K=0.1. ........................................ 14

2.7 The Huffman tree corresponding to Table 2.1 on page 18. ............... 19

2.8 Implementation of the LZ Coder in Python. ........................... 20

2.9 Implementation of the LZ Decoder in Python. .......................... 21

2.10 Pseudocode of the Arithmetic Coder ................................. 21

2.11 Pseudocode of the Arithmetic Decoder ................................ 22

2.12 Lossless JPEG causal template. ..................................... 26

2.13 The causal template for JPEG-LS .................................... 28

3.1 The causal template used by the predictor. ............................. 34

3.2 Template vs Predictor Performance as measured by the Second Norm of the Residual. ........................................ 35

3.3 Matlab code building and solving the overdetermined system of equations used to find the predictor coefficients. .................... 35

3.4 The listing of the GSL program used to solve the overdetermined system $\mathbf{Ax} = \mathbf{b}$. ........................................... 38

3.5 A causal template for a toy example. Traditionally x marks the sample to be predicted (in ancient times it has also been used to show the buried treasure). ........................................ 39
ACKNOWLEDGMENT

I would like to thank my advisor Dr. Wm Randolph Franklin and current and past members of our research group including but not limited to Dr. Barbara Cutler, Dr. Frank Luk, Dr. Marcus Vinicius Andrade, Zhongyi Xie, Jonathan Muckell, Jake Stookey and Dan Tracy. I also would like to thank members of my doctoral committee Dr. Michael Wozny and Dr. George Nagy. Dr. Nagy was kind enough to play a few chess games with me, when the weather was right and he was not too busy. I also would like to thank my wife Rabia Sarica and my family for their support through my graduate studies. Those are some of the people without whose support and generosity my thesis would not have been possible.
ABSTRACT

A novel compression scheme named Overdetermined Compression (ODETCOM) for 16 bit Digital Elevation Models (DEMs) is proposed. We achieve excellent lossless compression ratios, surpassing the competing image compression algorithms like variants of JPEG (JPEG-LS and JPEG 2000). We also achieve a very competitive near-lossless (lossy with error bounds) compression, often surpassing the compression of JPEG-LS by a large margin.

ODETCOM is based on a causal template model. We use a novel approach to train our model. An overdetermined system of linear equations is used to compute our model’s parameters. We minimize the second norm of the residual to strike a balance between the conflicting predictor requirements. The error between the prediction and the sample value is compressed. Using our method we manage to achieve compression rates consistently better than the lossless JPEG variants.
CHAPTER 1
INTRODUCTION

Humankind’s quest to map the Earth’s surface is still continuing in the 21st century; only the tools have changed. Nowadays we are equipped with computers, lasers, radars, planes and satellites. These technologies enable extremely fast and accurate topography acquisition. However, the problem of managing and manipulating this information remains.

One particular type of map is the elevation map; when stored on a computer, it is often called a Digital Elevation Model (DEM). A DEM is usually a square matrix of elevations. It is like an image, except that it contains a single channel of information: elevation; while a natural image would typically contain three channels of information: the primary colors red, green and blue. In that respect a DEM is more like an image containing various shades of gray.

This chapter aims to give an overall picture of the terrain compression problem.

1.1 Motivation

Elevation datasets are an ever expanding part of human knowledge. Current technologies allow for very fast elevation data acquisition. One such technology is LIDAR (Light Detection and Range), which is a laser based range detector coupled with a GPS sensor. LIDAR allows for 20,000 to 50,000 readings per second. Each reading is stored as an \((x, y, z)\) tuple where each coordinate is an IEEE double amounting to 24 Bpp (bytes per point). LIDAR is the technology used by the state of North Carolina after the Hurricane Floyd (1999); it was used to map the whole state in the NC Floodplain Mapping Project [17]. Just the Neuse river basin (11% of the whole NC area) is approximately 500 million points, requiring 11.2 GB.

Another fast data acquisition technology is IFSAR (Interferometric Synthetic Aperture Radar). SRTM (Shuttle Radar Topography Mission) used this technology to map 80% of the Earths landmasses, which amounts to approximately 120 million \(km^2\) [63], see Figure 1.1 on the following page. The shuttle Endeavour was launched
in February 2000, in a 57° inclined equatorial orbit, at 515 miles altitude. This was
the first accurate global scale digital topographic map. The horizontal resolution
of the maps released for civilian use was 90 m and the vertical error was less than
16 m. The amount of data collected in 11 days was in excess of 12 TB. This was
approaching the estimate for all the printed material in the Library of Congress at
the time. It was estimated that if all the printed matter in the Library of Congress
were stored as a plaintext, it would take between 15 TB to 20 TB to store [2].

One of the basic algorithms operating on DEMs is data compression. Com-
pressing a DEM reduces the storage space and may facilitate faster access to the
data as well as faster transfer. Generic compression algorithms are not very useful
on terrain datasets as they cannot exploit the 2D nature of the data and the natural
properties of the terrain dataset composition. Image compression algorithms
are relevant as they deal with similar issues. However, image compression algorithms
target natural images, where the information loss brought by imperfect reconstruc-
tion is not a big problem.

Many image compression algorithms alter their input to achieve better com-
pression rates. It is the human visual perception that is being targeted by the image
compression. Quantifiable differences between the input and the output are not im-
portant as long as the human eye perceives a similar enough image. This is not
acceptable when DEM is at stake. In fact, users of DEMs are very reluctant to work with altered datasets. Some issues are: altered slope, altered hydrology and altered visibility. Nevertheless, there is a real need for a terrain elevation product that has lower accuracy and is compact enough for field work.

Contrary to expectations, the ways we handle and store the elevation data have not advanced much. We still keep our data mostly as a grid of elevations and the compression applications used on these datasets do not perform well. The gzip program based on the Lempel-Ziv algorithm is used to compress the USGS DEMs (United States Geological Survey Digital Elevation Map) [61]. Below is a brief excerpt from the 00README file in the USGS DEM directory:

“The files have been compressed with the GNU “gzip” utility. If you do not have access to gzip, the FTP server will uncompress the file as you retrieve it...”

The Lempel-Ziv compression used in the gzip program is designed to compress sequential data [74]. The elevation datasets are known to have limited accuracy. The target accuracy for the SRTM mission was $\leq 16$ m absolute vertical accuracy, $\leq 10$ m relative vertical accuracy and $\leq 20$ m absolute horizontal circular accuracy at a 90% confidence level [45]. Unfortunately none of the terrain compression algorithms benefit from the allowable inaccuracy in the elevation datasets. Franklin et al. argue that a small elevation error in the coding process can be tolerated and exploited [20, 21]. Using a lossless compression algorithm instead of a lossy one, may unnecessarily constrict the compressor, resulting in suboptimal compression. Generic compression algorithms like gzip and bzip2 are all lossless. The image compression algorithm JPEG 2000 can achieve both lossless and lossy compression. While the lossy compression option of JPEG 2000 is superior to the ordinary JPEG compression, it is recognized as unsuitable for terrain datasets, as it harms subsequent data processing [40, 43]. There is a less known JPEG variant, JPEG-LS, which can provide an error bound on its output [67]. Algorithms like JPEG-LS, which can provide an error guarantee, are called near-lossless [31].
CHAPTER 2
RELATED WORK

In this chapter we discuss different terrain representations like contours (hypsography), elevation rasters (DEM) and Triangulated Irregular Networks (TIN). We also discuss different algorithms operating on those datasets, such as ODETLAP (Overdetermined Laplacian PDE), LOD (Level of Detail), generic compression algorithms, image compression algorithms and problems with those algorithms.

2.1 Terrain Representation

Compression of a terrain dataset depends on the model used to represent the dataset. It should be stressed again that this work is using rectangular, regularly sampled terrain elevation datasets like DEM (Digital Elevation Model). Among other representations there are irregularly sampled elevation points stored in a TIN (Triangulated Irregular Network) and the contour representation. Spatial data structures such as the quadtree of Samet [58] can also be used for terrain representation.

2.1.1 Contour Representation

Representing terrain with contour lines is perhaps the oldest representation; historic topographic maps use this representation. A contour line represents all of the points that are on the same elevation. A map consisting of contour lines is called a hypsogram. Hypsos is a Greek word meaning height and thus hypsography is literally a writing of heights. Hypsography is also the study of elevation distribution; for the distribution of elevations on the Earth see Figure 2.1 on the next page.

Contours have problems with flat areas like plateaus and plains. They look very good on paper and they are still being used since for some areas, no other representations exist. The digitized versions of contour maps are stored in the DLG (Digital Line Graph) format. A fine example of a topographic map of Crane Mountain near Thurman, NY can be seen in Figure 2.2 on page 6.
2.1.2 DEM Representation

The human eye cannot discern many shades of gray and an image is usually limited to 8 bpp (bits-per-pixel), resulting in 256 distinct shades of gray, where the 0th is black and 255th is pure white. Our planet’s elevation has considerably more variation than could be accounted for with 255 different levels of elevation. However, the number of levels also depend on the unit used to measure elevation. If miles are used, the vertical resolution of our DEM would be low and 8 bits would be more than enough. However, the resulting maps will be mostly flat. With miles used as an elevation measure, a New York map will be one flat plane with the same elevation level for both Mt. Marcy and Manhattan. On the other hand, if foot is used, then more digits are required to take into account the elevation of Mt. Everest. The fact that many of the elevation acquisition methods have much lower accuracy, does not justify the use of foot as a unit of measurement in DEMs. (There are still many digital and paper maps using foot as a unit of elevation measure.) A compromise
can be found in the metric system. The meter, which is a little longer than a yard is usually used as an elevation unit in digital maps in US and abroad. While 8 bpp (bits-per-pixel) cannot account for the topology variations on the Earth’s surface, 16 bpp is quite enough when the metric system is used.

Among DEM examples are: USGS DEM, NED (National Elevation Dataset), GTOPO30 (Global Topographic Data with 30′ horizontal resolution), DTED (Digital Terrain Elevation Data), BIL (Band Interleave by Line), SDTS (Spatial Data Transfer Standard). USGS DEMs of different scale provide coverage over the whole continental US. 1 : 250K scale DEMs are low fidelity datasets that are available in square datasets measuring 1201 × 1201 samples; their horizontal resolution is 3″. 1 : 24K scale DEMs on the other hand have 10 m horizontal resolution.

One problem is that it is difficult to edge match DEMs, owing to the shape of the geoid and data acquisition problems. DEMs for high latitude areas can have
rows with different number of samples. Small scale DEMs (low horizontal resolution) for the latitudes between 50° and 70° north have 601 samples per degree, while those north of 70° have 401 samples per degree. Similarly, adjacent DEMs acquired with different equipment may have jumps along edge boundaries and may need extensive editing to edge join.

DEM{}s have three different classification levels. Level-1 DEMs are reserved for datasets acquired from scanning high altitude photography. Those are usually reserved for large scale DEMs. The desired accuracy for those DEMs is RMSE (Root Mean Square Error) of 7m. The maximum permitted RMSE is 15m. Level-2 DEMs are the elevation datasets that have been processed (smoothed) and edited to remove systematic errors. They are typically derived from digitized contour maps. The maximum permitted RMSE for Level-2 DEMs is half contour interval. Level-3 DEMs are similarly derived from DLG (Digital Line Graphs) and incorporate hypsography (elevations) and hydrography (water bodies) data. They have a maximum permitted RMSE of one-third contour interval [62].

\[
RMSE = \sqrt{\frac{\sum(Z_i - Z_t)^2}{n}}
\]

- \(Z_i\) – interpolated DEM elevation of a test point
- \(Z_t\) – true elevation of a test point
- \(n\) – number of test points

A rendering of a 10 m horizontal resolution DEM of Kauai, Hawaii together with the touristic map of the same island can be seen in Figure 2.3 on the next page.

2.1.3 TIN

A TIN (Triangulated Irregular Network) is a collection of elevation samples (a point set) together with a triangulation, such that the terrain topography is approximated by non-overlapping planar triangles. The triangulation is defined in 2D but used in 3D. Thus the triangulation is defined on the projection of the elevation points on the \(x, y\) plane. Such a projection is easy to obtain by simply eliminating the \(z\) coordinate, which is later re-added for the representation.

A good TIN will contain more points in areas of higher variations such as
Figure 2.3: Kauai Touristic Map and Kauai DEM (rendered with MICRODEM [22]).
mountains, and fewer points in flat areas such as planes and plateaus. A TIN is ideally a set of points with edge relations among them. We can reduce a TIN to a set of points by bringing in a restriction on the triangulation. Among the different triangulation methods, one stands out: the Delaunay triangulation. The Delaunay triangulation is unique for a point set with arbitrary precision and thus allows a TIN to be stripped of its edge information.

A TIN is often one of the steps in the topography acquisition process. A point set acquired by a LIDAR system is often filtered for errors and converted into a TIN, which is next converted into a digital raster map.

The triangulation used is usually Delaunay, as the Delaunay triangulation has many nice properties. The Delaunay triangulation is named after Boris Delaunay, a Russian mathematician and mountain climber, who published a paper describing the triangulation in 1934 [14]. Boris Delaunay was a student of Georgy Voronoi, who is credited with the dual of the Delaunay triangulation, the Voronoi Diagram, published in 1907 [3, 65]. The concept however may have originated even earlier.

The Delaunay triangulation tends to have fewer “sliver” triangles and is defined to have The Empty Circle Property: the circumcircle of a Delaunay triangle does not contain any other points. It is easy to convert any triangulation into a Delaunay triangulation by flipping the common edges of pairs of adjacent triangles until The Empty Circle Property is satisfied. Other properties include being a planar graph and The Minimum Angle Property, which states that the minimum angle of each Delaunay triangle is as large as possible. Delaunay triangulation is also reducible to the convex hull problem in the next higher dimension.

Any Delaunay Triangulation with \( n \) vertices will contain at most \( 3n - 6 \) edges, which is the number of edges in a maximal planar graph. This linear bound on the number of edges has several implications. An algorithm that depends on the shortest pairwise distances between the points can be reduced from quadratic to linear complexity as a Delaunay triangulation will have an edge between every closest vertex pair. Another implication is that an algorithm that converts an arbitrary planar triangulation into a Delaunay triangulation has a linear complexity, since the number of the edges to be flipped is linear in the number of vertices, and every
A good example of a TIN close up generated with Franklin’s TIN algorithm, first implemented in 1973, can be seen in Figure 2.4. The first publication of the TIN idea was in the same year by Peuker et. al. [50]. Floriani et. al. applied the TIN idea to hierarchical surface representation [18, 19]. TIN also helps terrain visibility calculations [44]. A streaming computation of TIN was developed in 2006 by Isenburg, Liu, Shewchuk and Snoeyink [29].

2.1.3.1 Simulation of Simplicity

Simulation of Simplicity (SoS) is a technique that was developed in late eighties by Edelsbrunner and Mucke [16]. It is a general technique that can be used with all geometry algorithms to remove degeneracies. It works by consistently perturbing the data, in order to eliminate degeneracies. For example a vertex on a line segment is a degeneracy. An algorithm checking whether a point is inside a polygon or not, must resolve the degeneracy; a small perturbation of the vertex will move the point either inside or outside of the polygon. Similarly, in a Delaunay triangulation a square can be triangulated using either diagonal; in this case, a small perturbation will help to get a unique triangulation.
2.1.4 Related Techniques

A few techniques that are related to the compression problem are mentioned below. Those are Level of Detail (LOD), ODETLAP (Overdetermined Laplacian PDE) and computing terrain slopes. LOD is a problem that is relevant when visualizing large terrain datasets. ODETLAP is a technique which can be used to fill in gaps in the terrain elevation and approximate terrains from a point set. Terrain slopes and a method to compute terrain slopes is also presented.

2.1.4.1 Level of Detail

A problem related to the problem of terrain compression is the Level of Detail (LOD) problem. The LOD problem is defined on terrain datasets as the problem of managing complexity by removing elevation points where they do not matter much, such as those in flat areas. LOD is also extensively used by terrain visualization engines, where an observer has a limited view of the terrain. An open source terrain visualization engine is the VTP (Virtual Terrain Project) by Ben Discoe [15]. It uses an open source LOD implementation called libMini [55]. Roettger describes libMini in [56].

One of the earliest works on the problem of terrain visualization was done by NASA [23]. The problem was to visualize the surface of the planet Mars. There were 105 polygons, which at the time (1992-1993) were beyond the capabilities of the graphics hardware. The elevation data was organized in a quadtree, which kept patches of size either 10x10 or 5x5 elevations in the leafs of the tree. Every four siblings were joined into a parent node, which had one quarter of their resolution. The resolution level was picked by the user or it was adjusted for the area being viewed. This technology is still being used and can be observed in action in the NASA’s open source World Wind program [46]. The idea of using a quadtree for the task can be traced back to 1984 [39].

LOD also finds application in computer games. Bishop et al. describe an LOD system where the terrain is kept in pages each containing $65 \times 65$ elevations [4]. The terrain rendering process keeps only 30 to 40 of these pages in memory. The rule of thumb is to render the triangles so that each has approximately the same screen-
space-area. The Painter’s Algorithm, which is not specific for this application, is used for rendering by Bishop et. al. The distant pages are also more sparsely sampled than the pages closer to the viewpoint. The Painter’s Algorithm proceeds as a painter will do by painting the background first and proceeding with closer objects, which are painted over the background. Bob Ross, the painter from the public television series *The Joy of Painting*, was using this approach. The Painter’s Algorithm computes the hidden and visible parts of the objects in a 3D scene mapped to 2D, solving the Hidden Surface Removal Problem [48]. The Painter’s Algorithm is not without its own problems; it cannot resolve cases where objects alternately obscure one another.

With some exceptions, algorithms developed for LOD do not compress the terrain. One notable exception is the work of Losasso and Hoppe which uses compressed nested rectangular grids [37]. The compression starts by downsampling the terrain into a number of coarser representations. The coarse terrain is refined to generate a finer level and the difference between the terrain refined from a downsampled level and the one, which comes from a higher level is compressed. To compress the residual, the PTC (Progressive Transmission Coder) of Malvar is used [38].

### 2.1.4.2 ODETLAP

ODETLAP is based on an overdetermined system of Laplacian Partial Differential Equations (PDE). The Laplacian PDE is discretized on a mesh (Galerkin method); it is approximated by an averaging operator over the neighbors of every point on the mesh, represented by the following integral equation:

\[
z_{x,y} = \frac{1}{4} (z_{x-1,y} + z_{x,y-1} + z_{x+1,y} + z_{x,y+1})
\]

A typical application is a boundary value problem setting the function values on the boundary; the result approximates a continuous function at every one of the mesh points. In our application we do not have boundaries, instead we know the values attained at some of the mesh points. For those values we add corresponding equations to the linear system; constructing an overdetermined linear system.

A regular Laplacian PDE will result in a function that cannot assume a local
maximum at an interior point. The discretization of the Laplacian PDE can give us an intuition. Every point in the interior is the average of its neighbors. Thus no point can attain a maximum value. This is sometimes called *The Maximum Principle*. In contrast, ODETLAP can be used to infer maximum values in the gaps of elevation data, see Figure 2.5.

ODETLAP has an extra parameter $K$ that can be controlled; it is the constant multiplier for the equations representing known elevations; usually $K = 1$. However, different weights can be assigned to achieve a smoother approximation. Larger values, such as $K = 10$, tend to over-smooth the results, while smaller values such as $K = 0.1$ preserve sharp features.

ODETLAP is notable for its decent results when filling missing elevation data. A range of experiments were conducted to compare ODETLAP to the algorithms in the Matlab software package [28]; see Figure 2.6 on the following page for the result of an experiment.

ODETLAP can also be used to approximate the original terrain from from a
Figure 2.6: Filling a circular hole (R=40) of missing data using ODETLAP with K=0.1.

point set of sparsely allocated elevations. One of the ways of getting a good point set is to use the points from a Triangulated Irregular Network (TIN), as the fitness of the approximation depends on the number of points and their dispersion throughout terrain. However, a terrain model based on ODETLAP cannot limit the error of the representation.

When ODETLAP is used on a TIN, it acts as a low pass filter. The combination of TIN and ODETLAP can be used as terrain representation method. This terrain representation is very compact and can serve as a compression technique. One minor obstacle is that while in a DEM \((x, y)\) coordinates of the elevation points are implicit and only the matrix of the \(z\) values (elevation) is stored, with a point set there is a need to store the \((x, y)\) coordinates as well. A solution is to use delta coding on the \((x, y)\) coordinates. A good representation can be obtained if the Trav-
eling Salesman Problem is first run to order the \((x, y)\) coordinates on the shortest closed route.

2.1.4.3 Terrain Slope

Assuming that the surface is modeled by a continuous function \(Z\), the magnitude of the gradient at a specific point is defined as the slope \(A\) [27, 69].

\[
A = ||\nabla Z||_2
\]

In reality, we do not have a differentiable function \(Z\) to work with. If we have a DEM, all we have are many sample points on a regular grid. One of the methods that lets us calculate the slope on DEMs is the Zevenbergen-Thorne method or the ZT method [73]. The ZT method was suggested in 1987 and ever since it has remained one of the best methods to estimate the terrain slope [27, 33]. The ZT method estimates the slope using the four elevation samples at the four cardinal directions: \(N\), \(S\), \(E\) and \(W\).

\[
G = (E - W)/2\Delta x
\]
\[
H = (N - S)/2\Delta x
\]
\[
A = \sqrt{(G^2 + H^2)}
\]
\[
A^\circ = \arctan(A)
\]

The slope is an important terrain derivative. A misrepresented elevation can severely disrupt the slope computation. If all the elevations are off by at most one unit, the values of \(G\) and \(H\) can be off by \(1/\Delta x\) and this difference will be amplified when \(G\) and \(H\) are taken to the second power. However, having an error bound is much better than not. With an error bound one can tabulate the maximum slope error as a function of the error. Unfortunately an inverse function for this relation is not known.

Yet another issue is the horizontal resolution of the data. Hunter argues that 30 m DEMs can represent the slope with precision no better than 1.9° [27]. Simi-
larly, Kienzle states that resolutions worse than 25 m cannot represent steep slopes correctly [33].

2.2 Context Dilution

Compressing terrain elevation datasets with generic tools like gzip has yet another shortcoming. Terrain elevation samples are stored as 16 bit integers. Generic compression applications like gzip treat the data as a string of 8 bit integers. This has been recognized as a problem in the compression of Unicode strings, which are also comprised of 16 bit integers. A solution for the Unicode is to use SCSU (Standard Compression Scheme for Unicode) [60]. From Unicode FAQ:

“Q: What’s wrong with using standard compression algorithms such as Huffman coding or patent-free variants of LZW?

A: SCSU bridges the gap between an 8 bit based LZW and a 16 bit encoded Unicode text, by removing the extra redundancy that is part of the coding (sequences of every other byte being the same) and not a redundancy in the content. The output of SCSU should be sent to LZW for block compression where that’s desired.

To get the same effect with one of the popular general purpose algorithms, like Huffman or any of the variants of Lempel-Ziv compression, it would have to be retargeted to 16 bit, losing effectiveness due to the larger alphabet size. It’s relatively easy to work out the math for the Huffman case to show how many extra bits the compressed text would need just because the alphabet was larger. Similar effects exist for LZW. For a detailed discussion of general text compression issues see the book Text Compression by Bell, Cleary and Witten (Prentice Hall 1990)…”

Not unlike Unicode, the redundancy in elevation samples may appear in every other byte as the high byte is less likely to change over a small range of samples. The Unicode problem statement also hints at an altogether different problem: the problem of modeling 16 bit data. Producing statistics on 16 bit data is harder since the alphabet size is larger and the resulting statistics are sparse. This problem has
many names. It has been called “the context dilution problem” and in the data compression community it is often known as “the sparse context problem” [66].

2.3 The Modern Compression Paradigm

The modern compression paradigm divides the compression into two phases. The first phase is the data modeling. The second phase is the coding of the model [12]. The model feeds the coder with predictions and the input is encoded with reference to this prediction. Rissanen and Langdon argue that the problem of coding with respect to the model is already solved [53]. Indeed, Arithmetic Coding is optimal when supplied with correct probabilities [25]. If the probabilities provided by the model are flawed, Arithmetic Coding can result in a suboptimal coding where the average length of the output diverges from the ideal model’s entropy [5]. The method of arithmetic coding is attributed to Risannen and dates back to 1976 [52]. Witten, Neal and Cleary mention Abramson’s book from 1963 that gives credit to Elias for the concept that became Arithmetic Coding [1, 71]. Witten et al. implement an Arithmetic Coder decoupled from the model; they use the C programming language for the implementation [71].

If the premise of the modern compression paradigm is valid, we need not concern ourselves with the process of coding and should concentrate our efforts on the problem of modeling our data. The compression is tightly bound to the model; hence the compression performance will be as good as the model. This proposal aims to provide a model for terrain elevation datasets, which may help us with a compression tailored particularly for the elevation datasets.

2.4 Generic Compression Algorithms

Generic compression algorithms are entropy reduction techniques that have been known for many years. The best one is the Arithmetic Coding; it is used together with a high order data model to compress the residuals of our ODET-COM method. The other compression algorithms are presented to give a historical perspective.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Huffman Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0.25</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>0.25</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 2.1: A Huffman code for three symbols A, B and C.

2.4.1 Huffman Coding

Huffman Coding is a redundancy coding scheme that assigns binary codewords proportional in length to the probability of the symbols they represent [26]. The relationship is that the codeword assigned is \( -\lceil \log_2(P_{\text{symbol}}) \rceil \) bits long. This makes it optimal within one bit. The codewords are assigned based on a binary tree that is built using a bottom up approach. The symbols make up the leaves and those with equal probabilities are combined into internal nodes with a probability equal to the sum of their leaves. The bit assignment is done on the branches. The tree is finished when all of the symbols stem from the same internal node that becomes the root of the tree. Codewords are then made from the bits encountered on the path from the root to the corresponding symbol. Since there is always a unique path from the root to a leaf node, the codewords are also unique. A typical Huffman tree will assign the same bit (one or zero) on all of the left branches and the opposite bit on all of the right branches. Either convention is fine as the aggregate probabilities of the left and right subtrees at any internal node are always equal. The purpose of a convention like that is to ease the implementation of the tree building algorithm. To decode Huffman coded data, the Huffman tree or equivalently the codeword assignment used for the coding has to be transmitted along the data. Table 2.1 shows a toy example of a Huffman codeword assignment. Figure 2.7 on the following page contains the corresponding Huffman tree.

An important property of the Huffman code is that it is prefix-free, no codeword is a prefix of another codeword. Thus there is no need for delimiters between codewords.
2.4.2 LZ Compression

Lempel-Ziv (LZ) compression dates back to 1977 and is named after its inventors [74]. In LZ compression the border between the modeling and coding is blurred. LZ is a dictionary construction method where variable length strings are represented by a constant length dictionary index. The dictionary is built on the fly adapting to the input. The idea behind LZ is simple. LZ starts with a dictionary containing all the possible symbols. As the input symbols are read in succession, a string is constructed. At a certain point, a symbol added to the compound string will generate a string not yet in the dictionary. At that point, the new string is added to the dictionary, and the index of the string without the trailing symbol is output. The algorithm continues with a fresh string containing only the trailing symbol from the previous step.

In this manner, often-occurring strings are represented by their far shorter dictionary indices, resulting in a significant compression. The weakness of LZ stems from the fact that it is slow to adapt. New dictionary entries can be at most one symbol longer than the already existing ones, making LZ a suboptimal compression method. LZ compression is easy to code in Python, as Python natively supports hash tables that make it easy to build a table of strings with a constant lookup factor. A simple implementation can be seen in Figure 2.8 on the next page.

The decoding process follows the coding process and is similarly easy to im-
def encLZ(a):
    # build a dictionary containing all the ASCII characters
    d = dict((chr(i), i) for i in range(256))
    count = 256  # the index of the next dictionary entry
    w = ''  # currently the string is empty
    out = []  # there is nothing to output yet
    for c in a:  # Reads the symbols in succession
        if w+c in d:  # if the compound string is already in the dictionary
            w += c  # add the trailing symbol to the dictionary
        else:
            d[w+c] = count  # at the compound string to the dictionary
            count += 1  # increment the index
            out += [d[w]]  # output the index of the string
            w = c  # start fresh with the trailing symbol
            out += [d[w]]  # flush the last string to the output
    return out  # return the output

Figure 2.8: Implementation of the LZ Coder in Python.

2.4.3 Arithmetic Coding

The problem of Huffman Coding is that it is optimal only when all of the probabilities are a power of two. A symbol with probability $P = 0.3$ has an optimal codeword assignment that is a fractional number of bits, which cannot be accommodated by Huffman Coding. Arithmetic Coding is a redundancy coding method that solves this problem [25, 52, 71].

Arithmetic Coding reduces a stream of symbols to a fraction between zero and one; representing that fraction requires arbitrary precision but there are ways to circumvent this requirement.

Every symbol is assigned a range on a scale from zero to one, which is proportional to the probability of the symbol. Starting from the initial range $[0,1)$, the coder refines that range with the encounter of each additional symbol. After the last symbol, any fraction within the resulting range can be output. The pseudocode can be seen in Figure 2.10 on the next page.

The Arithmetic Decoder similarly maintains a table of the symbol ranges and
def decLZ(x):
    # build a dictionary of the ASCII symbols
    d = dict((i,chr(i)) for i in range(256))
    count = 256 # index of the next entry
    out = [ d[x[0]] ] # add the first symbol to the output
    w = d[x[0]] # start the string
    for index in x[1:]: # iterate over all the indices
        if index < 256: # if the index is in the ASCII table
            entry = d[index] # the entry is an ASCII symbol
        else:
            if index in d: # if the entry is in the dictionary
                entry = d[index] # we have a string
            else: # the index is not in the dictionary
                entry = w+w[0] # we end with the symbol we start
        out += [entry] # add the entry to the output
        d[count] = w + entry[0] # add a new dictionary entry
        count += 1 # increment the index
        w = entry # start with a new string
    return ''.join(out) # concatenate all of the strings

Figure 2.9: Implementation of the LZ Decoder in Python.

function Refine(range, symbol)
    low = range.low + symbol.low * (range.high - range.low)
    high = range.low + symbol.high * (range.high - range.low)
    return [low, high)

x.range = [0, 1)
while Input
    symbol = getFromInput()
    x.range = Refine(x.range, symbol.range)
return x.range.low

Figure 2.10: Pseudocode of the Arithmetic Coder
function FindSymbol(X)
    Find Symbol S such that:
        S.range.low <= X < S.range.high
    return S
do
    symbol = FindSymbol(X)
    print symbol
    X = X - symbol.range.low
    X = X / (symbol.range.high - symbol.range.low)
until symbol == Terminal Symbol

Figure 2.11: Pseudocode of the Arithmetic Decoder

given a fraction it figures out the range it belongs to, and produces the symbol corresponding to that range. The fraction is further modified to reverse the operation of the coder and the operation is repeated until the input stream is fully restored. To be able to stop, the decoder must either know the number of symbols to be decoded or there should be a symbol demarcating the end of the stream. Typically the latter option is chosen. The pseudocode of the Arithmetic Decoder is in Figure 2.11.

As it is, Arithmetic Coder/Decoder is trivial to implement using any arbitrary precision system such as the UNIX calculator language bc [11]. However, our limited experiments show that the optimization level of the arbitrary precision arithmetic on modern compilers is not up to the challenge. An implementation in the bc language, takes from 10 to 15 minutes to encode a string of 160,000 symbols. Experiments with arbitrary precision libraries available for other languages show that they are more suitable for accounting than for Arithmetic Coder implementation.

Some Arithmetic Coder implementations circumvent the problem of arbitrary precision by rescaling to 16 bit integer arithmetic. The following is observed: as the refinement process of the coder works on the symbols in the input stream, only a few digits are affected. During the range calculations the high bits of `range.low` and `range.high` converge and convergent bits are no longer affected by subsequent calculations. Thus one can maintain two 16 bit registers for the `range.low` and `range.high` and shift their matching high bits to the left then shift in the new bits from the right.
This scheme is prone to underflow that happens when the bit patterns of the low and high registers approach 01... and 10... respectively. A weak analogy can be made: infinite precision numbers do not have a unique representation, 0.1000... is the same as 0.0111.... After underflow subsequent operations may not cause convergence within the 16 bits of the working window. It can be resolved by successively removing the second bit from both of the registers until the high bits converge. Meanwhile, one has to keep track of the number of the bits removed. As soon as the high bits converge, the bit opposite to the convergent bit is shifted to the left (output), as many times as the number of bits removed. We effectively enlarge the 16 bit working window by keeping track of the count.

2.4.4 High-Order Modeling

A symbol-for-symbol code cannot account for the higher order statistics that may exist in the data. To further eliminate the redundancy we can use techniques that group symbols together. One such technique is the Lempel-Ziv compression. It is relatively easy to implement but it is suboptimal. We can also use off-the-shelf solutions, like general purpose compression utilities gzip and bzip2 to capture higher order statistics and further eliminate redundancy in the input data.

Higher order statistics are best illustrated by letter probabilities in the English language as in Shannon’s 1951 paper [59].

If 0-order statistics are used we can only tell the frequencies of different letters. When we move up to 1-order statistics we start taking into account the preceding letter as well and we keep statistics of digrams. With second order statistics we compute the frequencies of the trigrams and so forth. If the symbols in the input stream are dependent, high order statistics will help us to exploit that dependency. For example the letter “u” will often follow the letter “q” in English and a coder can exploit this fact by assigning a short codeword to the “qu” digram.

The disadvantage of using high order statistics is that the memory requirements are higher. When using bytes a 0-order approach will require a table of size $2^8$, while the 1-order approach will need $2^{16}$ table entries, 2-order approach $2^{24}$ entries and so forth. Using high order statistics does not automatically result in higher
2.5 Image Compression

The problem of image compression using causal models is defined as the problem of maximizing:

\[ P(x^n) = \Pi_{i=0}^{n-1} p(x_{i+1} | x^i) \]

where \( x^i = x_0x_1...x_i \) is the set of pixels encountered so far. \( p(x_{i+1} | x^i) \) is the conditional probability of encountering \( x_{i+1} \) after having encountered \( x^i \). This is where the name “causal model” comes from.

Causal models are not the only way of compressing an image. Subband Coding is also very popular. Subband Coding involves a transform that carries the image to the frequency domain and then encodes the image as a collection of different frequency bands.

Terrain compression is related to image compression. The terrain dataset is a rectangular dataset of elevation samples. Similarly an image is a rectangular dataset of pixels.

Any image compression algorithm can be applied to terrain datasets. One difficulty is that the terrain dataset must first be converted to an image file format and there are not many image formats that allow 16 bpp.

The most popular and the most successful image compression algorithms are in the JPEG family. Among those, notable are the ordinary JPEG (includes JPEG-DCT and Lossless JPEG), JPEG-LS and JPEG 2000. The names of the standards, JPEG-DCT, JPEG 2000 and so forth, serve a twofold role: they indicate both the image format and the compression algorithm used by the method. We will be mostly dealing with the compression algorithm as the image format is not relevant.

There are many other image compression algorithms that are also relevant. A popular one is PNG, which is an open source GIF replacement and enjoys widespread browser compatibility. Others like TMW and FELICS are not that popular and are usually not implemented beyond the research community. Those image formats will be discussed further down the text.
2.5.1 The JPEG Family

The ordinary JPEG (JPEG-DCT) is one of the most popular image formats on the Internet. The standard defining JPEG is ITU (CCITT) T.81 and emerged in September 1992 [9]. It is based on the Discrete Cosine Transform (DCT), which is related to the Fast Fourier Transform (FFT). It provides excellent compression rates for natural images. However, JPEG tends to smudge images having a limited color pallet and images containing sharp discontinuities. For this reason, JPEG is generally not used for images containing text. Text images and computer generated images that contain limited number of colors are often compressed with PNG and other document image formats. For a discussion on document image formats, see Section 2.6 on page 30.

JPEG-DCT partitions the image into blocks, where typically a block would contain 8 × 8 pixels. The block is treated as a matrix of numbers and is subjected to DCT. DCT is a transform that would carry the block of pixels from the spatial to the frequency domain. It is very similar to the Fourier Transform but only the even components (cosines) are used and no complex numbers are ever involved. Not using the sine components eliminates the imaginary part of the equation:

\[ e^{i\theta} = \cos \theta + i \sin \theta \]

The JPEG standard defines forward DCT (FDCT) used to encode images and inverse DCT (IDCT) used to decode images.

**FDCT** :

\[
S_{uv} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos \left(\frac{2x+1}{16} u\pi\right) \cos \left(\frac{2y+1}{16} v\pi\right)
\]

**IDCT** :

\[
s_{yx} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} s_{uy} C_u C_v S_{uv} \cos \left(\frac{2x+1}{16} u\pi\right) \cos \left(\frac{2y+1}{16} v\pi\right)
\]

\[
C_u, C_v = \frac{1}{\sqrt{2}} \quad \text{for} \quad u, v = 0
\]

\[
C_u, C_v = 1 \quad \text{otherwise}
\]

The relevant property of natural images is that a few frequency components carry most of the energy of the image. This means that if the result of the DCT is
Table 2.2: Lossless JPEG predictors based on the samples $a$, $b$ and $c$ of the causal template in Figure 2.12.

<table>
<thead>
<tr>
<th>Selection Value</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Prediction</td>
</tr>
<tr>
<td>1</td>
<td>$P_x = R_a$</td>
</tr>
<tr>
<td>2</td>
<td>$P_x = R_b$</td>
</tr>
<tr>
<td>3</td>
<td>$P_x = R_c$</td>
</tr>
<tr>
<td>4</td>
<td>$P_x = R_a + R_b - R_c$</td>
</tr>
<tr>
<td>5</td>
<td>$P_x = R_a + (R_b - R_c)/2$</td>
</tr>
<tr>
<td>6</td>
<td>$P_x = R_b + (R_a - R_c)/2$</td>
</tr>
<tr>
<td>7</td>
<td>$(R_a + R_b)/2$</td>
</tr>
</tbody>
</table>

Figure 2.12: Lossless JPEG causal template.

truncated and quantized, we can get a good approximation of the image when we apply the inverse DCT transform. It is often argued that DCT is a good approximation to the Karhunen-Loeve Transform (KLT), which is a fancy name for the eigenvalue (spectral) decomposition [32, 36]. Using KLT, we can retain only a few of the largest eigenvalues and their corresponding eigenvectors and use them to reconstruct data with adequate fidelity. Yet another name for the same technique is Principal Component Analysis (PCA), which is often used to find the spread of the data. The argument for use of DCT instead of KLT is that eigenvalue decomposition is numerically intensive and numerically unstable.

Lossless JPEG is part of standard ITU T.81 to which JPEG-DCT belongs [9]. However Lossless JPEG is a different mode and is not based on DCT. It is a prediction based scheme, where the new pixel is predicted using a causal template of three pixels, see Figure 2.12. There are eight different predictors to select from, see Table 2.2. The error is encoded and compressed using either Huffman or Arithmetic
compression. This scheme has a very low compression performance, typically 2:1 on color images [34]. JPEG implementations never include Lossless JPEG, as it is now rendered obsolete by JPEG-LS and JPEG 2000 Lossless. Even before the standardization of JPEG-LS, Lossless JPEG was outperformed by the PNG standard [34].

JPEG-LS originated in 1997 [31]. It is not a popular format. It has a lossless and a near-lossless mode. The term near-lossless refers to the ability of JPEG-LS to bound the error of the reconstruction. JPEG-LS defines a parameter called NEAR that controls the maximum error allowed in the representation. NEAR is zero for the lossless mode and non-zero for the near-lossless mode. For example, with NEAR = 2, the original pixel value at (x, y), will be at most two units off the reconstructed pixel value at the same location. The lossless and the near-lossless modes of the JPEG-LS use the same algorithm with a different NEAR parameter. JPEG-LS uses a causal template of size four, see Figure 2.13 on the following page. The predictor relies on only three of the pixels in the causal template:

\[
\tilde{x}_{med} = \begin{cases} 
min(a, b) & \text{if } c \geq \max(a, b) \\
\max(a, b) & \text{if } c \leq \min(a, b) \\
a + b - c & \text{otherwise}
\end{cases}
\]

This is a simple predictor that can generate large residuals. To correct for bias in the error a further step is used. The local gradient is approximated by \(g_1 = d - b\), \(g_2 = b - c\) and \(g_3 = c - a\). The quantized values of \(g_1\), \(g_2\) and \(g_3\) are used to determine the context and predict the error and correct for a possible bias. This is in contrast to a typical image compression algorithm that establishes context based on the previous errors. JPEG-LS uses only a few pixels in its predictor, which leads to a mediocre predictor performance.

JPEG-LS and other similar algorithms often employ a shortcut predictor. This shortcut predictor is called run-mode in JPEG-LS. It is enacted when there is not much to predict. A flat area is where the JPEG-LS’ gradient will indicate that there is not much change neither in the horizontal nor in the vertical direction. In such a case a simple counter is used to count the number of the elevations which are level.
Figure 2.13: The causal template for JPEG-LS.

JPEG 2000 is based on an entirely different algorithm. Following the success of the wavelet based compression for the FBI fingerprint database [7] in 1993 and the SPIHT algorithm by Said and Pearlman [57] in 1996, JPEG 2000 adopted a wavelet based compression scheme that became a standard in December 2000. Two different wavelet transforms are used for the two distinctly different modes of the algorithm. One of the transforms is an integer based transform, which is reversible and used for the lossless mode of the algorithm. The other transform is floating point based and irreversible, hence used for the lossy mode of the algorithm. The result of the floating point transform is also further quantized for an even higher compression ratio. The coefficients resulting from the wavelet transform are compressed using an Arithmetic Coder that operates on the bitplanes of the coefficients using a layered approach.

2.5.2 Other Image Compression Techniques

Other image compression techniques include PNG, TMW and FELICS. Those are good examples for different approaches to image compression. PNG (Portable Network Graphics) is a lossless image compression algorithm that has replaced GIF (Graphics Interchange Format) [54]. It is notable for its pre-processing filters. One of the filters simply subtracts the current pixel from the previous pixel. The result of the filter can be interpreted as the error of the prediction. A homogeneous area of the image will result in a string of zeros, which are easier to compress. This simple technique can lead to considerable improvement in the compression performance.

TMW, an unexplained acronym, which probably stands for the names of the authors Tischer, Meyer and a third person or a pet, is yet another image processing technique that uses a number of linear predictors that are blended together for a
better result [42]. The linear predictors are obtained from an optimization that is run during a static image analysis. TMW is notable for a very large number of parameters as well as for multiple predictors that are blended together.

There are a few TMW extensions in the literature [41, 72]. One such extension improves the performance of TMW by using a Least Squares technique to minimize the error of prediction [72]. While this approach reduces the complexity of TMW, the results reported are superior to the original TMW. However, there are couple of flaws. The predictor is trained around a single pixel and its performance depends on that pixel’s neighborhood. If the neighborhood does not reflect the statistics of the rest of the image then the predictor will be flawed. Another problem is that the setup of the linear system, described in this work, does not include the constant factor, often called DC, which will reduce the effectiveness of the regression.

FELICS (Fast and Efficient Lossless Image Compression) is an image compression algorithm that has a very simple predictor [24]. Its predictor is based on only two pixels. On the first row those are the pixels preceding the pixel to be predicted $(1, j - 2)$ and $(1, j - 1)$. On any other row, those are the pixel on the left $(i, j - 1)$ and the upper pixel $(i - 1, j)$. The lower of the two predictor pixels is called L and the higher is called H. A single bit is used to indicate whether the upcoming pixel is between L and H or is outside. Another bit is used to indicate whether the pixel outside is above H or below L. Finally, the error is encoded. Of course, this simple scheme cannot keep up with more complicated ones. However, its first row predictor is not bad, considering the lack of image statistics at the start.

### 2.5.3 Medical Images and High Dynamic Range Images

Medical images carry some similarities with DEM data. Medical images have a bit-depth higher than 8 bpp (bits-per-pixel). A typical Radiology image is 12 bpp [66]. Because of the high depth they also suffer from sparse statistics like the terrain datasets. As discussed before, sparse statistics hamper the performance of the compression algorithms.

Another similarity is that they need to be compressed either losslessly or with performance guarantees. The issues of medical liability dictate this requirement and
lossy image manipulation techniques do not qualify. Those high bit depths are the reason that some of the popular image compression algorithms support bit depths of up to 16 bpp.

It has long been known to the photographers that a printed color picture is not nearly as realistic as a slide projection. One of the many differences is that a slide projection has a higher contrast ratio [68]. The term contrast ratio has entered our lives in the past decade with the introduction of digital TV technologies. The contrast ratio reflects the difference between the dark and light colors an apparatus can reproduce. High Dynamic Range (HDR) images cater to the fact that standard RGB pictures composed of the primary colors red, green and blue are not good enough to represent high contrast ratios [70]. An HDR image can show both the dark colors and the light colors simultaneously. An ordinary camera can obtain a good rendition of the dark colors with an over-exposure. In such case the light colors would be washed out. Similarly an under-exposure will render a good range of the light colors while rendering all of the dark colors black. In the absence of an HDR sensor HDR images are composed from the pictures taken over a range of exposures [13]. To better represent luminance in those pictures an extra channel of information often 16 or 32 bits deep is used [70].

2.6 Document Compression

Document compression is another area related to terrain compression. Printed matter is one of the most durable media used to contain the human knowledge. While many digital media are hampered by the lack of hardware to read them, books have been used for centuries. Digitization of documents requires document sensitive compression algorithms. The challenge is that there are many repeating patterns (letters) and there may be graphics that share the pages. Some algorithms that address those challenges are CCITT Group 3, CCITT Group 4, JBIG2 and DjVu.

CCITT Group 3 and Group 4 are standards, used by fax machines, developed in the early 80s [8, 10]. Those compression algorithms compress bi-level images as those resulting from scanning documents and printed matter. Group 3 is purely
1-D, every scanline is compressed on its own. Group 4 is more sophisticated: every scanline is compressed with reference to the previous one. Adjacent scanlines in documents are often similar. This property allows Group 4 to achieve a considerably higher compression than Group 3 at the cost of higher complexity. The TIFF image format has Group 4 as a compression option.

JBIG2 is another compression format that provides compression for bi-level images [30]. It uses causal template predictors of varying sizes. Smaller templates of 10 pixels are used for small images like icons and letters. Larger templates of size 13 pixels are used for larger images. Repeating patterns like letters are collected into a symbol dictionary and reused by specifying only locations on the page. JBIG2 can also handle images as its templates can be used on dithering patterns. Finally a binary Arithmetic Coder is used for the entropy coding.

DjVu is a proprietary format by LizardTech Inc. [35]. It segments the documents into images (background) and letters (foreground) and applies different compression algorithms to those. Letters are compressed with an algorithm dubbed JB2, which is similar to JBIG2. The continuous tone images on the other hand are compressed with a subband coding scheme using a wavelet called IW44 this; algorithm bears similarity to the algorithm used in JPEG 2000. The entropy coding algorithm used in DjVu is the ZP-Coder. Details are available on the Z-Coder based on which ZP-Coder is built [6]. Z-Coder is an adaptive Rice coder [51]. It claims vastly improved compression over JPEG but some of its algorithms are trade secrets.
CHAPTER 3
THE ODETCOM METHOD

We describe the overdetermined compression (ODETCOM) and how it works. We start with the ODETCOM model, the predictor, the training of the predictor and we exemplify this on a toy dataset. We also describe different ways of coding the error. And finally we extend our model to handle the near-lossless compression.

A note on notation: Traditionally bold capital letters are used for matrices, such as $A$, and bold lower case letters are used for vectors, such as $b$. Similarly monospaced fonts are used for computer code, such as `while b==2`. In the following text we will follow this convention with the addition that the causal template samples as well as the computer code variables holding their values are also typeset in monospace. There is a distinction between the vector $b$ and the elevation sample $b$. Similarly the causal templates traditionally mark the position of the next sample with an $x$, which has nothing to do with the vector $x$. Since computer codes do not allow bold entries, we will use $m$ to denote a matrix variable and $v$ for a vector. Thus $A_m$ holds a matrix and $b_v$ holds a vector. Scalar values are traditionally italicized.

3.1 The ODETCOM Model

The compression model consists of a predictor algorithm and a module to encode and, if possible, further compress the residual. The predictor algorithm is a causal template model that derives its parameters in a novel way; it uses an overdetermined linear equation system. The causal template model is used to predict upcoming elevations in a raster order. The difference between the prediction and the real elevation is the error/residual. For near-lossless coding we further quantize the residual into what we call buckets. After bucketing we have two options to entropy code the quantized residual: Arithmetic Coding with high-order error modeling and unary coding followed by an off-the-shelf entropy reduction algorithm such as bzip2. Arithmetic Coding is slower but it is the better option and that is what we use.
3.1.1 The ODETCOM Predictor

A causal template model looks into the neighborhood of the target elevation to be predicted and infers its elevation. Since the elevations are in a raster order, the predictor is effectively extrapolating the next elevation from the already available ones. This is due to the spatial coherence of adjacent elevation samples. Different terrain types (hilly, mountainous, high or low resolution) have a quantifiably different relation between the neighboring elevations.

We use a causal template of size eight, see Figure 3.1 on the next page. It is a linear predictor that is trained using multiple pixels, possibly the whole image. The training process uses an overdetermined system of linear equations \( Ax = b \), that finds a compromise for the best set of the coefficients \( x \). Each equation corresponds to a prediction and consists of the eight elevations in the causal template and a constant term. The right-hand-side is a vector \( b \) of the elevations that we are trying to predict. The unknown vector of coefficients \( x \) can be solved for, using a linear system solver capable of solving overdetermined systems. No exact solution exists for such a system, unless the number of linearly independent rows matches the number of unknowns, which is unlikely; there is always a residual vector: \( r = Ax - b \). A solution that minimizes the second norm of the residual maximizes the utility of the predictor. We use Matlab’s generic solver that invokes a QR decomposition algorithm on the matrix \( A \), resulting in the much easier to solve \( QRx = b \). \( Q \) is an orthogonal matrix and thus \( QT = Q^{-1} \). Both sides of the \( QRx = b \) equation can be multiplied by \( QT \) to eliminate it from the left-hand-side, resulting in \( Rx = QTb \). The coefficient vector \( x \) can be solved for using back-substitution, since the system \( Rx = QTb \) is triangular. Eventually the coefficient vector \( x \) minimizes the second norm of the residual vector \( r \).

This method for building the predictor has the added benefit that it is extremely easy to evaluate the effectiveness of different size causal templates. The accuracy of the predictor exhibits diminishing returns as we increase the size of the causal template. The accuracy of the predictor is important but the model must not be too heavy either. The interplay between accuracy and the size of the model is used to find a good trade off.
Figure 3.1: The causal template used by the predictor.

Our template size of eight is not arbitrary. Using our algorithm we can train our predictor using smaller size templates and continuously add new template positions until the performance does not improve significantly. It is reasonable to start with a template including only a few samples in the near vicinity of the prediction. We start with only three samples b, c, and d, using the naming scheme in Figure 3.1. We gradually expand our template size and for each template, we train and test our predictor. We can construct a new matrix A and vector b from the test data and use the x computed on the training data to get a new residual. We plot the second norm of the residual obtained from the test data versus various templates, see Figure 3.2 on the next page. With the increase in the template size, we observe diminishing returns.

3.1.2 Computing the Predictor Coefficients

It is relatively easy to compute the predictor coefficients using the Matlab programming language. The resulting code is short and easy to read. The Figure 3.3 on the following page shows a snippet of Matlab code used to find the predictor coefficients. The code keeps terrain elevations in the variable ter, variables a, b, c, d, e, f, g, and h contain the samples caught under the causal template, see Figure 3.1. x is the sample to be predicted.

\[ A_m \text{ and } b_v \text{ are respectively the matrix on the lhs of the linear system } Ax = b \text{ and the vector on the rhs. Line 11 of the code builds the matrix concatenating individual rows, where each row includes a constant factor. The rhs of the linear system contains the values of the elevations to be predicted; they are concatenated into a single column vector in Line 13. The system is overdetermined as nine linearly} \]
Figure 3.2: Template vs Predictor Performance as measured by the Second Norm of the Residual.

```
1: for i=3:400
2:   for j=3:100
3:     a = ter(i,j-2);
4:     b = ter(i,j-1);
5:     c = ter(i-1,j-1);
6:     d = ter(i-1,j);
7:     e = ter(i-1,j+1);
8:     f = ter(i-1,j+2);
9:     g = ter(i-2,j);
10:    h = ter(i-2,j+1);
11:    A_m = [A_m; a b c d e f g h 1];
12:    x = ter(i,j);
13:    b_v = [b_v; x];
14:   end
15: end
16: x_v = A_m_v;
```

Figure 3.3: Matlab code building and solving the overdetermined system of equations used to find the predictor coefficients.
independent rows would have been enough to solve the system. In the code snippet in Figure 3.3 on the preceding page we build a matrix with 38,808 rows (we will address this number later).

An overdetermined system cannot be solved exactly. The solution to the system minimizes the residual $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$. Every row in the linear system corresponds to a different template position. Every entry in the residual vector corresponds to the error, predictor would make at that template position. Thus minimization of the residual increases the accuracy of the predictor.

There is nothing magical about the number 38,808. In this code snippet we use the first 100 columns of the terrain dataset to train the predictor. We do not use the template positions that have extraneous sample positions. For example, if we use the first row, the top two rows of the template will protrude. For this reason we start from the third row. Similarly, if we use the first two columns, a part of the template will protrude to the left. We end up using approximately 24% of the 160,000 samples in our dataset.

Matlab’s backslash operator picks an appropriate solver at Line 16 in Figure 3.3 on the previous page. The time to solve this system is 89 seconds on a T60 Thinkpad laptop with a 2 GHz CPU and 1 GB of RAM. We can speed up the program in two different ways: we can either use fewer elevation samples making the system lighter or we can use the C programming language. The first solution trades off the accuracy of the predictor for fast operation.

During experiments we find out that using only 7 rows, with approximately 2,800 total samples results in acceptable predictor performance and short running times (0.27 seconds).

The second solution involves using the C programming language with a linear algebra library. We pick GNU Scientific Library (GSL). Using GSL’s QR-decomposition function and subsequently solving the system using back-substitution, we achieve a speed-up of 240× over the original Matlab program (89 : 0.37), without sacrificing the predictor’s accuracy. The listing of the resulting program is short and can be seen in Figure 3.4 on page 38. Most of the C code is for memory management and data acquisition. Only two lines of code, the lines 26 and 27, do the actual work,
the rest support the computation.

In fact, the resulting C language version is so fast, that it can solve the system in under 0.5 seconds even if we use all of the elevation samples for training (except those on the dataset’s borders).

To predict values on the borders of the dataset we use a couple of different approaches. The very first two samples are stored as they are, no prediction is made. Following samples on the first row use a simple predictor based on the two previous samples. We call this predictor the first row predictor and its operation is described in Sub-section 3.1.3. For all other template positions on the border, if there are missing elevation samples, then we approximate them with the nearest sample in the causal template.

### 3.1.3 Toy Example

This example will use a causal template of size two, see Figure 3.5 on page 39. A similar template is used by FELICS image compression algorithm [24]. We will use a $5 \times 5$ terrain dataset $T$, that belongs to a larger terrain elevation dataset.

$$T = \begin{bmatrix}
279 & 290 & 305 & 323 & 336 \\
276 & 287 & 302 & 322 & 336 \\
275 & 285 & 299 & 319 & 334 \\
277 & 286 & 297 & 316 & 333 \\
286 & 295 & 301 & 314 & 328 
\end{bmatrix}$$

A rendering of the dataset with extreme low values in dark blue and extreme high values in dark red is in Figure 3.6 on page 39.

We will use the second and the third row to train the predictor. We will not use the data in the first column as the template will be protruding to the left. Similarly we avoid the first row as we will be using a simpler predictor for that row. There are eight samples in the second and the third row when we exclude the first column. Those samples are inside the rectangle in Figure 3.7 on page 39.

For all of the training samples, we position the causal template. The position of the causal template for the first sample can be seen in Figure 3.8 on page 40. We
1: #include <stdio.h>
2: #include <gsl/gsl_linalg.h>
3: #include <gsl/gsl_matrix.h>
4:
5: int main (void)
6: {
7:     FILE *fpm;
8:     FILE *fpv;
9:     fpm = fopen("A.txt","r");
10:    fpv = fopen("b.txt","r");
11:    if (!fpm || !fpv){
12:        fprintf(stderr,"Sorry. Could not open the files! ");
13:        exit(0);
14:    }
15:    gsl_matrix *A = gsl_matrix_alloc (38808,9);
16:    gsl_vector *b = gsl_vector_alloc (38808);
17:    gsl_vector *x = gsl_vector_alloc (9);
18:    gsl_vector *tau = gsl_vector_alloc (9);
19:    gsl_vector *r = gsl_vector_alloc (38808);
20:
21:    int sm = gsl_matrix_fscanf (fpm, A);
22:    int sv = gsl_vector_fscanf (fpv, b);
23:    fclose(fpm);
24:    fclose(fpv);
25:
26:    gsl_linalg_QR_decomp (A, tau);
27:    gsl_linalg_QR_lssolve (A, tau, b, x, r);
28:    gsl_vector_fprintf (stdout, x, "%g");
29:
30:    gsl_vector_free (x);
31:    gsl_vector_free (r);
32:    gsl_vector_free (b);
33:    gsl_vector_free (tau);
34:    gsl_matrix_free (A);
35:    return 0;
36: }

Figure 3.4: The listing of the GSL program used to solve the overdetermined system $Ax = b$. 

Figure 3.5: A causal template for a toy example. Traditionally x marks the sample to be predicted (in ancient times it has also been used to show the buried treasure).

Figure 3.6: A rendering of the terrain dataset T.

can see from Figure 3.8 on the following page that the template samples a and b acquire the elevation values 276 and 290 respectively. The elevation sample to be

$$T = \begin{bmatrix}
279 & 290 & 305 & 323 & 336 \\
276 & 287 & 302 & 322 & 336 \\
275 & 285 & 299 & 319 & 334 \\
277 & 286 & 297 & 316 & 333 \\
286 & 295 & 301 & 314 & 328
\end{bmatrix}$$

Figure 3.7: The samples used for training are inside the rectangle.
Figure 3.8: The position of the causal template on the first sample.

predicted is 287. We construct our linear system to train the predictor coefficients.

\[
T = \begin{bmatrix}
279 & 290 & 305 & 323 & 336 \\
276 & 287 & 302 & 322 & 336 \\
275 & 285 & 299 & 319 & 334 \\
277 & 286 & 297 & 316 & 333 \\
286 & 295 & 301 & 314 & 328 \\
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
287 \\
302 \\
322 \\
336 \\
285 \\
299 \\
319 \\
334 \\
\end{bmatrix}
\]

The solution of this linear system yields the coefficients \( x \).

\[
x = \begin{bmatrix}
0.1698 \\
0.8779 \\
-14.1448 \\
\end{bmatrix}
\]

We should mention that the training of the predictor is the same as the regression method in statistics. In the example above it corresponds to a least squares approximation.

Since we have solved for our model’s parameters we can start coding. For the first column we choose to approximate the samples that are not available to the causal template. Thus we approximate the sample \( a \) with the sample \( b \).
Figure 3.9: A predictor for the first row.

We will show how the predictor works on the fourth row, including the first column. For the fourth row we get the following residual:

\[
\begin{bmatrix}
275 & 275 & 1 \\
277 & 285 & 1 \\
286 & 299 & 1 \\
297 & 319 & 1 \\
316 & 334 & 1 \\
\end{bmatrix}
\begin{bmatrix}
0.1698 \\
0.8779 \\
-14.1448 \\
\end{bmatrix}
- 
\begin{bmatrix}
277 \\
286 \\
297 \\
316 \\
333 \\
\end{bmatrix}
= 
\begin{bmatrix}
-3.0327 \\
-3.7630 \\
-0.0955 \\
0.3301 \\
-0.2756 \\
\end{bmatrix}
\]

Note how the first column approximated the unknown value of a with b. Since we are dealing with integer quantities we can round to the nearest integer, getting:

\[
\begin{bmatrix}
-3 \\
-4 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]

For the first row we will use a different predictor, see Figure 3.9. We will not train this predictor. We will use the following simple approximation: 2b-a. Of course, the first two elevation samples must be transmitted without coding as the causal template of the first row predictor does not cover them. For the first row we get the prediction values 301, 320, 341. The prediction errors are −4, −3 and 5 respectively.

Finally to transmit the elevation dataset T without any loss, we need to transmit the following: the model parameters (assuming the causal template is known to...
The first two elevation samples 279 and 290 and the errors of the predictor, preferably entropy coded, should also be transmitted. With all this information, the receiving party will be able to recover the elevation dataset $T$.

### 3.1.4 Coding the Residual

A typical histogram of the residual values shows that the magnitude of the residual exhibits exponential decay, see Figure 3.10 on the following page. Most of the residuals are zero and the numbers in the histogram bins diminish at an exponential rate as we go further from zero. There are coding techniques that are proven optimal for exponentially decaying distributions. Two such codes are the Unary Code and the Indexed (Rice) Code [51]. We get good results using unary code combined with an OTS (off-the-shelf) compression program like $bzip2$. However, the best results are obtained using Arithmetic Coding combined with high order modeling of the error. We use an Arithmetic Coder based on 16 bit integer arithmetic as described in the article by Witten et al. [71]. We use a more modern implementation by Nelson described in [47]. For near-lossless coding we pre-process the residual using a bucketing method that is going to be explained.

#### 3.1.4.1 The Unary Code

The Unary Code is one of the simplest possible codes. A positive number $x$ is represented by $x$ ones followed by a zero. Thus a number $x$ is represented by $x + 1$ binary symbols.

#### 3.1.4.2 The Indexed (Rice) Code

The Indexed Code is a little bit more sophisticated. It is a hybrid of unary and binary coding. The first part is unary coded and it indicates the general neighborhood (the bin) of the number, the second part contains the offset in the bin and is binary coded. Indexed Coding involves a parameter which defines the size of the
Figure 3.10: The histogram of a typical ODETCOM residual exhibits exponential decay on both sides of the axis. The histogram is for a $400 \times 400$ dataset with 160K residuals.

bin, which is always a power of two. A bin of size one will reduce to the Unary Code, making the Indexed Code a superset of the Unary Code. A bin of size two will need an offset that is one bit long. The bin size and the offset are related; the offset needs $\log_2(binsize)$ bits. Thus the first, unary part, is indexing the bins and the second part is the offset within the bin. The number of bits needed by the offset affects the length of the representation. An exponentially decaying distribution with a large mean will benefit from a large offset, while a distribution with a small mean will be optimally represented with a small or zero offset, i.e. the Unary Code. A table of Indexed Coding for different offset sizes can give us an intuition into the process, Table 3.1 on the next page.
Table 3.1: Indexed (Rice) Coding with Different Offsets, Offset of 0 bits corresponds to Unary Coding

<table>
<thead>
<tr>
<th>Number</th>
<th>Offset 0 bits</th>
<th>Offset 1 bit</th>
<th>Offset 2 bits</th>
<th>Offset 4 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
<td>000</td>
<td>00000</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>01</td>
<td>001</td>
<td>00001</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
<td>100</td>
<td>010</td>
<td>00010</td>
</tr>
<tr>
<td>3</td>
<td>1110</td>
<td>101</td>
<td>011</td>
<td>00011</td>
</tr>
<tr>
<td>4</td>
<td>11110</td>
<td>1100</td>
<td>1000</td>
<td>00100</td>
</tr>
<tr>
<td>5</td>
<td>111110</td>
<td>11101</td>
<td>10101</td>
<td>00110</td>
</tr>
<tr>
<td>6</td>
<td>1111110</td>
<td>111100</td>
<td>110000</td>
<td>01000</td>
</tr>
<tr>
<td>7</td>
<td>11111110</td>
<td>1111101</td>
<td>111011</td>
<td>00111</td>
</tr>
<tr>
<td>8</td>
<td>111111110</td>
<td>11111100</td>
<td>111000</td>
<td>01000</td>
</tr>
</tbody>
</table>

3.1.4.3 Coding Negative Numbers

The Unary Code and the Indexed Code can only be used to encode positive numbers and zero. An extension is necessary to encode the negative numbers. One possible solution is to use a single bit prefix that is zero for positive and one for negative numbers. Another solution is to map all integers to natural numbers $\mathbb{Z} \rightarrow \mathbb{N}$. A simple and effective map is to map the negative integers to the odd natural numbers and positive integers to the even natural numbers. This mapping circumvents the need for a sign bit and sometimes results in a better compression ratio. One such map is the function:

$$f(x) = \begin{cases} 
-2x - 1 & x < 0 \\
2x & x \geq 0
\end{cases}$$

3.1.4.4 Near-Lossless Coding

Near-lossless coding can be implemented based on either Unary Coding or Indexed Coding. Given an error bound, a non-zero unary number can always be chopped to a shorter representation. For example, an error bound of 1 can be used to modify the unary representation of 4, which is 11110 to the one bit shorter 1110 (3), incurring a loss of one unit permissible given the error bound. Similarly using Indexed Coding, one can bucket different error values together and get rid of the
Table 3.2: Bucketing for near-lossless coding with error bound of 1.

<table>
<thead>
<tr>
<th>Number Mapped To</th>
<th>Bucket Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>-3</td>
</tr>
<tr>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

offsets. For example, an error bound of 1 and Indexed Coding with one bit offset will represent the number 3 as 101, if we get rid of the offset we get only the unary part: 10, which is shorter. Thus we will squash together the values of 2 and 3 as we can no longer differentiate between 100 and 101.

An even more powerful bucketing technique is possible. In the previous two examples we were rounding off numbers in a single direction, towards zero. We can do better than that and round off numbers in both directions. We can find a mapping that will map three distinct values to a single bin as opposed to having two distinct values mapped to a single bin when the error bound is 1. Our previous mapping was mapping 3 and 2 to 2 and 1 and 0 to 0. Our new mapping will map 2, 3 and 4 to 3. Both the right and the left neighbors on the number axis are considered, see Table 3.2.

If we define our error bound as \( \text{NEAR} \) we can find a \( \mathbb{Z}^+ \rightarrow \mathbb{Z}^+ \) function that can map any positive integer to its bucket index.

\[
f^+(x) = \left\lfloor \frac{x + \text{NEAR}}{2\text{NEAR} + 1} \right\rfloor
\]

Similarly for negative integers we can define:

\[
f^-(x) = \left\lfloor \frac{x - \text{NEAR}}{2\text{NEAR} + 1} \right\rfloor
\]
Using these two functions we can bucket the residuals and pass them straight to the Arithmetic Coder. However, care should be taken at the prediction stage, as the predictor should use the information available to the decoder. Otherwise the prediction in the coder and decoder would not be in sync. This is important only for the near-lossless coding as in the lossless mode there is no loss of information.

The bucketing strategy presented here is basically a quantization step that combines the symbols together allowing for higher compression rates. The NEAR parameter should be passed on to the decoder, together with other model parameters.
CHAPTER 4
RESULTS

In this chapter we present ODETCOM compression results of various datasets. We use terrain datasets with resolutions of 30 m, 10 m and 3 m. We also try extreme cases: random noise and a sea-level DEM. We compare our results to the compression performance of JPEG 2000 and JPEG-LS. We also apply ODETCOM to the popular cameraman image, which is 8 bpp and 256 × 256.

Datasets with different horizontal resolutions have different properties and it is reasonable to expect different compression ratios. It is also reasonable to expect that an algorithm that performs best on a certain dataset may not be the best choice for another.

For the training we use almost all of the elevation samples available for the dataset. We ignore those at the borders, where the causal template would protrude.

4.1 ODETCOM on 30 m Horizontal Resolution Datasets

We use 1000 DTED Level 2 datasets with 30 m horizontal resolution [64]. We standardize our datasets to contain 400 × 400 samples. We randomly pick 1000 datasets from the west coast of the USA, removing the ones that are all zero (sea level).

4.1.1 Lossless Results

We compare the compression performance of ODETCOM to that of JPEG 2000 (lossless mode) and JPEG-LS (NEAR = 0). We achieve a compression better than JPEG-LS on 966 out of 1000 datasets. Compared to JPEG 2000 we are also better; we are better on 907 out of 1000 datasets. When we compare against the best JPEG variant for each dataset, we are slightly worse, achieving better compression on only 900 out of 1000 datasets, see Table 4.1 on the next page. To see how much better we are, we look at the aggregate performance as well, see Table 4.2 on the following page.
Table 4.1: The breakdown on the lossless compression performance (30 m datasets).

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS vs. JPEG 2000</td>
<td>JPEG 2000 achieves better compression on 908/1000, 91% of the datasets</td>
</tr>
<tr>
<td>JPEG-LS vs. ODETCOM</td>
<td>ODETCOM achieves better compression on 966/1000, 97% of the datasets</td>
</tr>
<tr>
<td>JPEG 2000 vs. ODETCOM</td>
<td>ODETCOM achieves better compression on 907/1000, 91% of the datasets</td>
</tr>
<tr>
<td>Best JPEG vs. ODETCOM</td>
<td>ODETCOM achieves better compression on 900/1000, 90% of the datasets</td>
</tr>
</tbody>
</table>

Table 4.2: The aggregate performance of lossless ODETCOM and JPEG on 1000 30 m datasets.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression Ratio</th>
<th>Total Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS</td>
<td>6.91</td>
<td>46,343,203</td>
</tr>
<tr>
<td>JPEG 2000</td>
<td>7.62</td>
<td>41,994,761</td>
</tr>
<tr>
<td>ODETCOM</td>
<td>8.63</td>
<td>37,073,264</td>
</tr>
</tbody>
</table>

4.1.2 Near-Lossless Results

ODETCOM and JPEG-LS have a near-lossless mode that limits the maximum absolute error of the representation. The parameter that limits the maximum error is called NEAR in the JPEG-LS standard [31]. We use three different error limits: NEAR = 1, NEAR = 2 and NEAR = 4. With the increase in the allowed error the performance difference does not change much, see Table 4.3 on the next page. We also compare the compression ratios of ODETCOM and JPEG-LS for different values of NEAR, see Table 4.4 on the following page. We can see that the compression ratio on the 1000 ODETCOM compressed datasets is almost the same as that of JPEG-LS with a NEAR parameter twice that of ODETCOM.

4.2 ODETCOM on 10 m Horizontal Resolution Datasets

We use 430 large scale (1 : 24K) USGS DEM datasets with 10 m horizontal resolution [62]. We again standardize our datasets to contain 400 x 400 samples. We harvest our datasets from DEMs covering the following Hawaii islands: Kauai,
Table 4.3: The breakdown on the near-lossless compression performance (30 m datasets).

<table>
<thead>
<tr>
<th></th>
<th>JPEG-LS vs. ODETCOM (NEAR = 1)</th>
<th>JPEG-LS vs. ODETCOM (NEAR = 2)</th>
<th>JPEG-LS vs. ODETCOM (NEAR = 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODETCOM achieves better compression on 829/1000, 83% of the datasets</td>
<td></td>
<td>ODETCOM achieves better compression on 796/1000, 80% of the datasets</td>
<td>ODETCOM achieves better compression on 827/1000, 83% of the datasets</td>
</tr>
</tbody>
</table>

Table 4.4: The aggregate performance of near-lossless ODETCOM on 1000 30 m datasets.

<table>
<thead>
<tr>
<th>NEAR</th>
<th>Compression Ratios</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JPEG-LS</td>
</tr>
<tr>
<td>11.25</td>
<td>14.05</td>
</tr>
<tr>
<td>2</td>
<td>JPEG-LS</td>
</tr>
<tr>
<td>14.10</td>
<td>17.73</td>
</tr>
<tr>
<td>4</td>
<td>JPEG-LS</td>
</tr>
<tr>
<td>17.72</td>
<td>22.31</td>
</tr>
</tbody>
</table>

Lanai, Maui, Niihau and Oahu. We take care to ensure that none of our 430 datasets is all zero, since that would pollute the statistics.

4.2.1 Lossless Results

We compare our compression to that of JPEG 2000 and JPEG-LS. Lossless ODETCOM is better than JPEG 2000 in 415 out of 430 datasets. Similarly we are better than JPEG-LS in 421 out of 430 datasets, see Table 4.5 on the next page.

We also compare the aggregate compression on 430 datasets, see Table 4.6 on the following page.

4.2.2 Near-Lossless Results

We repeat the experiments we conducted on 30 m datasets. Again we use NEAR = 1, NEAR = 2 and NEAR = 4. We compare JPEG-LS and ODETCOM as JPEG 2000 does not have a near-lossless mode, see Table 4.7 and Table 4.8.
Table 4.5: The breakdown on the lossless compression performance (10 m datasets).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression Ratio</th>
<th>Total Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS vs. JPEG 2000</td>
<td>10.10</td>
<td>13,629,077</td>
</tr>
<tr>
<td>JPEG 2000</td>
<td>10.87</td>
<td>12,653,344</td>
</tr>
<tr>
<td>ODETCOM</td>
<td>12.74</td>
<td>10,803,906</td>
</tr>
</tbody>
</table>

Table 4.6: The aggregate performance of lossless ODETCOM and JPEG on 430 10 m datasets.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression Ratio</th>
<th>Total Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS</td>
<td>10.10</td>
<td>13,629,077</td>
</tr>
<tr>
<td>JPEG 2000</td>
<td>10.87</td>
<td>12,653,344</td>
</tr>
<tr>
<td>ODETCOM</td>
<td>12.74</td>
<td>10,803,906</td>
</tr>
</tbody>
</table>

Table 4.7: The breakdown on the near-lossless compression performance (10 m datasets).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression Ratio</th>
<th>Total Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS vs. ODETCOM (NEAR = 1)</td>
<td>15.60</td>
<td>21.27</td>
</tr>
<tr>
<td>ODETCOM</td>
<td>18.76</td>
<td>27.30</td>
</tr>
<tr>
<td>JPEG-LS vs. ODETCOM (NEAR = 2)</td>
<td>23.40</td>
<td>35.54</td>
</tr>
</tbody>
</table>

Table 4.8: The aggregate performance of near-lossless ODETCOM on 430 10 m datasets.

<table>
<thead>
<tr>
<th>NEAR</th>
<th>JPEG-LS</th>
<th>ODETCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15.60</td>
<td>21.27</td>
</tr>
<tr>
<td>2</td>
<td>18.76</td>
<td>27.30</td>
</tr>
<tr>
<td>4</td>
<td>23.40</td>
<td>35.54</td>
</tr>
</tbody>
</table>
Table 4.9: The breakdown on the lossless compression performance (3 m datasets).

<table>
<thead>
<tr>
<th>Algorithm Comparison</th>
<th>Breakdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS vs. JPEG 2000</td>
<td>JPEG 2000 achieves better compression on 60/72, 83% of the datasets</td>
</tr>
<tr>
<td>JPEG-LS vs. ODETCOM</td>
<td>ODETCOM achieves better compression on 48/72, 67% of the datasets</td>
</tr>
<tr>
<td>JPEG 2000 vs. ODETCOM</td>
<td>ODETCOM achieves better compression on 41/72, 57% of the datasets</td>
</tr>
<tr>
<td>Best JPEG vs. ODETCOM</td>
<td>ODETCOM achieves better compression on 36/72, 50% of the datasets</td>
</tr>
</tbody>
</table>

Table 4.10: The aggregate performance of lossless ODETCOM and JPEG on 72 3 m datasets.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression Ratio</th>
<th>Total Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS</td>
<td>20.28</td>
<td>1,135,963</td>
</tr>
<tr>
<td>JPEG 2000</td>
<td>21.11</td>
<td>1,091,561</td>
</tr>
<tr>
<td>ODETCOM</td>
<td>21.43</td>
<td>1,074,892</td>
</tr>
</tbody>
</table>

4.3 ODETCOM on 3 m Horizontal Resolution Datasets

We use 72 datasets with 3 m horizontal resolution. We standardize our datasets to contain 400 × 400 samples. Datasets were obtained from Marcus Andrade and belong to a region close to Renton, WA. The reason for having such a small set is that high resolution elevation datasets are scarce.

4.3.1 Lossless Results

We compare the compression performance of ODETCOM to that of JPEG 2000 (lossless mode) and JPEG-LS (NEAR = 0). We achieve a compression better than JPEG-LS on 48 out of 72 datasets. Compared to JPEG 2000, we are better on 41 out of 72 datasets. When we compare against the best JPEG variant we are even, achieving better compression on only 50% of the datasets, see Table 4.9. To see how much better we are, we look at the aggregate performance as well, see Table 4.10.
Table 4.11: The breakdown on the near-lossless compression performance (3 m datasets).

<table>
<thead>
<tr>
<th>JPEG-LS vs. ODETCOM (NEAR = 1)</th>
<th>ODETCOM achieves better compression on 26/72, 36% of the datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS vs. ODETCOM (NEAR = 2)</td>
<td>ODETCOM achieves better compression on 19/72, 26% of the datasets</td>
</tr>
<tr>
<td>JPEG-LS vs. ODETCOM (NEAR = 4)</td>
<td>ODETCOM achieves better compression on 17/72, 24% of the datasets</td>
</tr>
</tbody>
</table>

Table 4.12: The aggregate performance of near-lossless ODETCOM on 72 3 m datasets.

<table>
<thead>
<tr>
<th>Compression Ratios</th>
<th>NEAR</th>
<th>JPEG-LS</th>
<th>ODETCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34.49</td>
<td>29.12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>48.88</td>
<td>38.85</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>77.88</td>
<td>58.59</td>
<td></td>
</tr>
</tbody>
</table>

4.3.2 Near-Lossless Results

We use three different error limits: NEAR = 1, NEAR = 2 and NEAR = 4, see Table 4.11. We also compare the compression ratios of ODETCOM and JPEG-LS for different values of NEAR, see Table 4.12. The near-lossless performance of ODETCOM does not match that of JPEG-LS on our 3 m datasets. Nevertheless, the compression ratios are impressive. With high compression ratios and 400 × 400 datasets small changes in the file size can result in large compression ratio differences. For example, 100 Bytes compressed to 2 Bytes will result in a compression ratio of 50, if we manage to compress 100 Bytes to 1 Byte the 1 Byte difference doubles the compression ratio from 50 to 100.

4.4 Extreme Cases

We are also interested in ODETCOM’s performance in extreme cases such as encounter of noise and all-zero datasets. Real data often has noise and terrain datasets may cover the sea.
Table 4.13: The performance of lossless ODETCOM and JPEG on the all-zero dataset.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression Ratio (for 2 Bytes/sample)</th>
<th>Total Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS</td>
<td>3809.52</td>
<td>84</td>
</tr>
<tr>
<td>JPEG 2000</td>
<td>2133.33</td>
<td>150</td>
</tr>
<tr>
<td>ODETCOM</td>
<td>1333.33</td>
<td>240</td>
</tr>
</tbody>
</table>

Table 4.14: The performance of lossless ODETCOM and JPEG on the uniform noise dataset.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression Ratio</th>
<th>Ideal</th>
<th>Total Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS</td>
<td>1.49</td>
<td>1.6</td>
<td>214,334</td>
</tr>
<tr>
<td>JPEG 2000</td>
<td>1.49</td>
<td>1.6</td>
<td>215,459</td>
</tr>
<tr>
<td>ODETCOM</td>
<td>1.29</td>
<td>1.6</td>
<td>247,441</td>
</tr>
</tbody>
</table>

4.4.1 Compressing an All-Zero Dataset

We generate an all-zero 400 × 400 dataset. We compress it with ODETCOM and JPEG variants. Table 4.13 shows the size and compression achieved by lossless algorithms. Experiments on the all-zero dataset with near-lossless compression algorithms show that regardless of the NEAR parameter the compression is always the same as in the lossless variant, i.e. there is no difference between the lossless and near-lossless performance for this dataset. ODETCOM’s lower performance on the all-zero dataset indicates that the entropy coder used by ODETCOM is not as good as that of the JPEG variants.

4.4.2 Compressing Noise

We generate a 400 × 400 dataset consisting of integer elevations uniformly distributed between 1000 and 2000. We apply ODETCOM and the JPEG variants on the resulting dataset, see Table 4.14 and also Table 4.15 on the next page. As expected random data is difficult to compress. Ideally, we should have been able to compress the dataset down to 200,000 Bytes, a compression ratio of 1.6, since $-\log_2(1/1000) \approx 10$ bits and $400 \times 400 \times 10/8 = 200,000$ Bytes. We are slightly worse than the JPEG variants with the exception that we achieve a better compression ratio for NEAR = 4.
Table 4.15: The performance of near-lossless ODETCOM on the uniform noise dataset.

<table>
<thead>
<tr>
<th>Compression Ratio</th>
<th>NEAR</th>
<th>JPEG-LS</th>
<th>ODETCOM</th>
<th>Ideal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.76</td>
<td>1.40</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.90</td>
<td>1.84</td>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.11</td>
<td>2.34</td>
<td>2.4</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.1: The cameraman image, 256 × 256.

4.5 Application on Images

Although never intended for images, we test ODETCOM on the popular cameraman/surveyor image, see Figure 4.1. We compare the resulting lossless and near-lossless bitrates to the results obtained with JPEG-LS and JPEG 2000, see Table 4.16 on the next page. For this particular image we are doing slightly worse than the JPEG variants.
Table 4.16: Bitrate and compression ratio for the cameraman image
ODETCOM vs. JPEG variants.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Bitrate (bpp)</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG-LS (NEAR = 0)</td>
<td>4.31</td>
<td>1.85</td>
</tr>
<tr>
<td>JPEG 2000 (Lossless)</td>
<td>4.54</td>
<td>1.76</td>
</tr>
<tr>
<td>ODETCOM (NEAR = 0)</td>
<td>4.82</td>
<td>1.66</td>
</tr>
<tr>
<td>JPEG-LS (NEAR = 1)</td>
<td>2.86</td>
<td>2.80</td>
</tr>
<tr>
<td>ODETCOM (NEAR = 1)</td>
<td>3.13</td>
<td>2.55</td>
</tr>
<tr>
<td>JPEG-LS (NEAR = 2)</td>
<td>2.29</td>
<td>3.50</td>
</tr>
<tr>
<td>ODETCOM (NEAR = 2)</td>
<td>2.49</td>
<td>3.21</td>
</tr>
<tr>
<td>JPEG-LS (NEAR = 4)</td>
<td>1.63</td>
<td>4.90</td>
</tr>
<tr>
<td>ODETCOM (NEAR = 4)</td>
<td>1.86</td>
<td>4.30</td>
</tr>
</tbody>
</table>
CHAPTER 5
CONCLUSION AND FUTURE WORK

We have a novel terrain elevation modeling approach. It provides great lossless compression ratios and very competitive near-lossless coding, often surpassing JPEG variants.

Future work involves having an adaptive causal template, where the shape and the size of the causal template can be adapted to the dataset. In such cases the shape and size of the causal template should track the dataset.

It is also worth studying high fidelity elevation datasets; however, obtaining such datasets is difficult. Instead, we could retarget our application for medical images and perhaps HDR images. It will be interesting to see whether our idea will be applicable to such images.

As evidenced by the compression performance on the all-zero dataset, the entropy coder used by ODETCOM is not on par with the entropy coder in the JPEG variants, which is expected, as the implementation of the Arithmetic Coder used to compress the residuals is from 1991 [47]. It is worth exploring Adaptive Binary Arithmetic Coders that are extremely fast as they use only shifts and no multiplications; an early example is in [49]. They can be used to compress the bitplanes of the residuals, the way JPEG 2000 compresses wavelet coefficients. This may yield better compression performance and faster operation as well.
LITERATURE CITED


[70] _____, *High dynamic range imaging*,


6 Masters Theses

Evaluating and compressing hydrology on simplified terrain – Jon Muckell 802

Representation, compression and progressive transmission of digital terrain data using over-determined Laplacian partial differential equations – Zhongyi Xie 843

Parallel terrain compression and reconstruction – Jake Stookey 880
EVALUATING AND COMpressING HYDROLOGY ON SIMPLIFIED TERRAIN

By

Jonathan Muckell

A Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of MASTER OF SCIENCE

Major Subject: COMPUTER SYSTEMS ENGINEERING

Approved:

W. Randolph Franklin, Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

April 2008
(For Graduation May 2008)
CONTENTS

LIST OF TABLES ............................................................... iii
LIST OF FIGURES ............................................................... iv
ACKNOWLEDGMENT .............................................................. vi
ABSTRACT ................................................................. vii

1. Introduction ............................................................... 1
   1.1 Trends in Terrain Data and Hydrology .............................. 1
   1.2 Prior Art .............................................................. 1
   1.3 Overview ............................................................. 3

2. Measuring Hydrology Error ............................................... 4
   2.1 Drainage Network Error Metric ...................................... 4
   2.2 Ridge-River Drainage Calculation ................................... 5
      2.2.1 Assigning Flow Directions to Plateaus using Connected Com-
            ponents ............................................................ 9

3. Hydrology-Aware Terrain Simplification ................................. 13
   3.1 Terrain Simplification to Preserve the Hydrology ............... 13
      3.1.1 Approximating Terrain using Over-determined Laplacian PDEs 14
      3.1.2 ODETLAP Point Selection .................................... 15
      3.1.3 Recovering Terrain Using a ODETLAP Hydrology Customiza-
            tion .............................................................. 16
   3.2 Effectiveness of Ridge-Ridge ODETLAP Simplification .......... 17

4. Hydrology-Aware Triangulation of Terrain ............................... 20
   4.1 Prior GIS Triangulations ............................................ 20
   4.2 HydroTIN Algorithm ................................................ 22
   4.3 Results ............................................................... 25

5. Hydrology-Conditioned Datasets ......................................... 28
   5.1 Introduction .......................................................... 28
   5.2 Results ............................................................... 28

CONCLUSIONS .............................................................. 29
LIST OF TABLES

3.1 The amount of potential energy error for six 400 by 400 datasets sampled at 30m resolution. The percentage of flow traveling uphill is also shown, along with the compression ratio of each dataset. . . . . . . . . . . . . . 18
LIST OF FIGURES

2.1 To compute the potential energy error, the drainage is computed on the reconstructed terrain. This drainage network is then mapped onto the original terrain. The amount of water flowing uphill and downhill influences the metric. ............................................ 4

2.2 The ridge-river network, with rivers in black and ridges in white. ....... 6

2.3 (a) Our method for computing the drainage networks compared to (b) ArcGIS. Notice how our method is less fragmented then ArcGIS. ....... 6

2.4 Time to initialize and solve the sparse matrix (Matrix $A$) for small and large datasets. ................................................................. 9

2.5 (a) Drainage network(white) before handling flat regions. (b) Drainage network (black) and watershed boundary (white) after accounting for flat regions. ......................................................... 10

2.6 The connected components approach can also be used to determine watersheds. This image shows an isolated watershed with the boundary shown in black. ......................................................... 11

3.1 Flow chart of the ridge-river technique. Inputs are in boxes and programs in circles. ................................................................. 13

3.2 Simplifying the original drainage network using Douglas-Peucker. The refined line network is reduced by a factor of 3 with little visible difference. 16

3.3 Modifying the ODETLAP equations to better represent ridges and rivers has a drastic decrease in the amount of hydrology error. Both plotted lines above use the same set of points. .............................. 17

3.4 The images show the a 400 $\times$ 400 hill2 dataset sampled at 30m resolution and compressed using the ridge-river technique. The color regions represent the elevations with blue being low and red corresponding to high elevation. The black region shows the significant drainage network above the threshold of 100. The higher potential energy error metric correlates with a visible difference in the drainage network. Notice how the high error corresponds to short fragmented drainage networks. .... 19

4.1 HydroTIN is a targeted simplification technique that incorporates the structure of the ridge and rivers and optimized for hydrology preservation. ................................................................. 20
4.2

4.3 Left: Shows ridge-river points refined using Douglas-Peucker. This is the initial "seeding" of the triangulation. Right: Triangulation derived from the nodes on the left. Notice how the edges align with the ridge and river networks.

4.4 Forbidden zone

4.5 Amount of potential energy error determined when mapping the flow direction matrix from the reconstructed terrain onto the original elevation matrix. Error is weighted by the gradient and the amount of flow.

4.6 Percentage of cells that are flowing uphill when mapping the flow direction matrix obtained from the terrain reconstruction onto the original elevation matrix.

5.1 Low coupled, two stage process for computing drainage networks to avoid sampling and data collection issues.

5.2 LEFT: Original Terrain with watersheds (white) and drainage network (black). The flow here is blocked by a small ridge that occurs due to sampling and data collection errors. RIGHT: Flow passes through small insignificant barriers. Allowing larger and more realistic watersheds and drainage networks

5.3 The watersheds(white) and drainage networks(black) on the original and recovered terrains.
ACKNOWLEDGMENT

This research project would not have been possible without the support of many people. Words can not express my appreciation for these individuals for encouraging me to think innovatively, as well as providing deep insight to the working of the scientific community. I would like to express my gratitude to my advisor, Prof. Franklin who was abundantly helpful and offered invaluable assistance, support and guidance. Deepest gratitude towards Prof. Marcus Andrade and Prof. Barb Cutler for providing great ideas, motivation and insight in the areas of hydrology and computer graphics.

During this work I have collaborated with many colleagues for whom I have great regard, and I wish to extend my warmest thanks to all those who have helped me with my work. Special thanks also to all my graduate friends, especially group members; Prof. Frank Luk, Zhongyi Xie, Metin Inanc, Dan Tracy, and Jake Stookey for sharing the literature and for valuable discussions on terrain representation. I would also like to convey thanks to the Department of Electrical and Computer Systems Engineering and Faculty for providing financial support and laboratory facilities.

Thanks to the various governmental agencies for funding this research, which was supported by NSF grants CCR-0306502 and DMS-0327634, by DARPA/DSO/GeoStar, and by CNPq - the Brazilian Council of Technological and Scientific Development.
ABSTRACT

We present a metric based on the potential energy of water flow to determine the error introduced by terrain simplification algorithms. Typically, terrain compression algorithms seek to minimize RMS (root mean square) and maximum error. These metrics fail to capture whether a reconstructed terrain preserves the drainage network. A quantitative measurement of how accurately a drainage network captures the hydrology is very important for determining the effectiveness of a terrain simplification technique. Having a measurement for testing and comparing different models has the potential to be widely used in numerous applications (floods, erosion, pollutants, etc). In this paper, we first define a metric that maps the reconstructed drainage network onto the original terrain and computes the amount of energy needed for the water to flow. Two novel terrain simplification algorithms are presented that use a targeted compression to preserve the important hydrology features. These methods and other simplification schemes are then evaluated using the potential energy error metric to determine how much hydrology information is lost using the different compression techniques.
1. Introduction

1.1 Trends in Terrain Data and Hydrology

Terrain data is being sampled at ever increasing resolutions over larger geographic areas requiring special compression techniques to manipulate the data. Typically the effectiveness of a terrain compression technique is how well it minimizes the root mean square or the maximum error between the original terrain and the reconstructed geometry [7]. This metric is not always the best choice for preserving hydrological information, since channels and ridges, essential for the calculation of drainage networks [16], might be lost. For example, a scheme which naively interpolates the terrain between two points on opposite sides of a river can flatten the terrain and block flow.

Often measuring the amount of water flow occurs by taking ground truth measurements, where hydrology statistics are determined by direct measurement. This can be expensive, time consuming and requires accessing remote locations. Rapid technological advances are making it possible to have accurate, high resolution elevation data. This provides for a more accurate simulation of hydrology, in ways that were once impractical. In order for this to happen, it is essential that the scientific community has the tools available that can store and manipulate large terrain datasets [1]. Accurate hydrological simulations could allow better understanding of regions at greatest risk of flooding, help minimize the threat of natural disasters and to track and predict the flow of pollutants. This work could also be applied to other flow based models. For instance, instead of water, it could be used to understand threat areas due to volcanic activity. Also, it could be applied to high resolution data for segmentation based on the watersheds.

1.2 Prior Art

Past work has been done for defining a metric for comparing how well a computed drainage compares to the real world drainage [17]. The D8 model can assign flow in one of the eight possible directions. In the SFD (single flow direction) version
of the D8 model the entire amount of flow from each cell is entirely distributed to the lowest adjacent neighbor. This is not the case in the MFD (multi-flow direction) version the flow is fractionally distributed to all the lower adjacent neighbors.

A slightly more sophisticated MFD model is the $D_\infty$ model. As the name indicates, flow can travel in an infinite number of directions and is not limited to eight directions. The amount of water leaving each cell is distributed to one or more adjacent cells based on the steepest downward gradient [13].

Another implementation for finding drainage networks are digital elevation model networks or the DEMON model [4]. Instead of modeling flow as a point source that flows to an adjacent neighbor, DEMON captures the flow by contributing and dispersal areas. The motivation for using a method such as DEMON is that the representation allows for flow width to vary over nonplanar topography. However, this can introduce loops and inconsistencies in the hydrology.

Based on the fact that elevation data is only an approximation for the actual terrain, some methods allow for water to flow uphill until spilling over an edge. These flooding methods determine spill points out of every basin. In the Terraflow approach [5, 15], the path of least energy is used to flow uphill until reaching the spill point. The flow runs uphill in situations when there is not an adjacent lower elevation. These methods often keep expanding the drainage networks until they flow off the edge of the terrain. This is because it is assumed that the initial input DEM is prone to collection and sampling errors that cause unrealistic depressions.

The main benefits of Terraflow are the ability to avoid dataset issues, obtain long continuous river flow and scalability on massive datasets. The main disadvantages are that this approach may miss realistic drainage basins and poorer performance on non-massive datasets.

Typically for any of the method listed above, the inputs are a DEM (Digital Elevation Model) and a flow accumulation threshold. The outputs are a flow direction grid and a flow accumulation grid. The flow direction grid specifies the direction of flow. The flow accumulation grid is an integer corresponding to the amount of flow and a cell is considered part of the drainage network if its flow accumulation threshold is larger then the threshold value given as an input.
1.3 Overview

The contributions to this research are as follows:

1. A new metric for measuring the amount of hydrology error introduced by a terrain simplification algorithm. The drainage network is computed on the reconstructed terrain and then is mapped onto the original terrain. The amount of potential energy error is computed. Flow traveling uphill will increase the error, while flow traveling downhill will lower the amount of error.

2. Efficient drainage network computation based on a system of linear equations. The resulting drainage often contains longer and more realistic drainage networks than ArcGIS [11] which is typically regarded as the industry standard.

3. Simple, fast and effective computation of the ridge network. Inverting the terrain and running the drainage network provides an approximation of the ridge network. Often compression techniques smooth out ridges. Having an accurate representation of the ridge network can assist compression algorithms and also has applications in siting, path planning and hydrology.

4. Introduction of a new compression method that specifically minimizes hydrology error by using over-determined LaPlacian PDEs.

5. Triangulation algorithms for simplifying terrain are widely used in practice. A variant of a Triangulated Irregular Network called HydroTIN is described that is focused on minimizing the amount of potential energy error.

6. A method for removing insignificant ridges that unrealistically block water flow. These small ridges arise due to dataset and collection errors and are troublesome when executing a drainage network program. By eliminating these regions, longer, more naturally representative rivers result from the drainage network simulation.
2. Measuring Hydrology Error

2.1 Drainage Network Error Metric

Standard metrics for evaluating the effectiveness of terrain simplification algorithms use root mean squared (RMS) and maximum error. These measurements are ineffective when evaluating the loss of drainage network structure. Therefore one of the most important aspects of this paper is to introduce a metric geared towards measuring this error.

It is important to remember that the goal of our hydrology metric is not to compare the reconstructed hydrology against an absolute truth. Hydrology computed on a digital representation may have significant errors due to sampling and data collection inaccuracies. Therefore, our hydrology metric does not compare the reconstructed drainage network versus the original drainage network directly, as with ground-truth methods. Instead, the hydrology metric takes the flow direction grid and the flow accumulation grid from the reconstructed drainage and maps it onto the original DEM (Figure 2.1).

![Figure 2.1](image)

**Figure 2.1**: To compute the potential energy error, the drainage is computed on the reconstructed terrain. This drainage network is then mapped onto the original terrain. The amount of water flowing uphill and downhill influences the metric.

To compute the accuracy of the drainage network, the gradient, amount of flow contributing cells and whether the flow is traveling uphill or downhill are taken into account. The total downward energy and upward energy is computed as a
summation of the gradient $|(E_i - E_r)|$, where $E_i$ is the original elevation matrix and $E_r$ is the receiving elevation matrix where each cell contains the elevation of the adjacent cell in $E_i$ that is receiving the water flow. The gradient is weighted by the amount of flow (variable $W$). The final Error is determined as the ratio of the total upward energy divided by the total downward energy.

$$Energy_{Down} = \sum (E_i - E_r) \cdot W_i$$

$$Energy_{UP} = \sum (E_r - E_i) \cdot W_i$$

$$Error = \frac{Energy_{UP}}{Energy_{Down}}$$

In order to compute the energy error metric the flow is computed on the reconstructed DEM. The error is determined by comparing the flow direction matrix computed on the reconstructed geometry with the elevation matrix from the original DEM. A perfect match would have an energy matrix equal to zero. This would occur if the flow never went uphill, which is the case when using the flow direction grid from the original terrain. Therefore, the closer the metric is to zero, the more accurate the reconstructed drainage network.

### 2.2 Ridge-River Drainage Calculation

In this work, the drainage network is computed using a standard D8 model [13] based on steepest descent flow. In this implementation each cell flows to the lowest adjacent neighbor and flow is forbidden from traveling uphill. The method is executed on both the original and inverted terrain, and this can be done in parallel. The inverted terrain is derived from the original elevation matrix using the equation below:

$$I_e = Max(E) - E + Min(E) \quad (2.1)$$

where $E$ is the original elevation matrix and $I_e$ is the inverted elevation matrix. The drainage network is computed using $E$ to determine the drainage network and $I_e$ to determine the ridge network. The two networks are combined together and throughout this paper they will be referred to as the ridge-river network, as seen
in Figure 2.2. The process for computing each network is identical except the previously defined elevation inversion described above. Figure 2.3 shows our drainage network computation compared to ArcGIS developed by ESRI. Our implementation results in less fragmented and more realistic river networks.

![Figure 2.2: The ridge-river network, with rivers in black and ridges in white.](image)

![Figure 2.3: (a) Our method for computing the drainage networks compared to (b) ArcGIS. Notice how our method is less fragmented then ArcGIS.](image)

The output of the initial drainage computation is a flow accumulation grid, where each cell contains an integer corresponding to how many other cells contribute flow to that point. Cells above a predefined threshold are considered significant and are added to the drainage and ridge network, which we call the ridge-river network[10]. It is not necessary to store all these cells since they are clustered together and therefore add little value to a point selection compression technique. The
Douglas-Peucker [6] algorithm is used to reduce the number of points required to represent each river segment. The refined points can be stored and further compressed. To reconstruct the terrain we use an implementation of Over-determined Laplacian Partial Differential Equations (ODETLAP) [7] where each point is considered to be the average of its four neighbors, with a subset of the ridge-river points being known.

Different from other methods that use flooding [1], our method computes flow using a system of linear equations $Ax = b$ where $x$ is an unknown $N^2$ length vector equal to the amount of water accumulation at each cell and $b$ is the initial flow or “rain” at each cell, usually equal to 1. Matrix $A$ is a $N^2 \times N^2$ sparse matrix, where each non-zero element corresponds to cells that contribute flow to each cell. For instance, if cell $X_1$ receives flow from cell $X_2$ and $X_5$, row 1 in matrix $A$ will contain non-zero elements in columns 1, 2 and 5. Therefore the number of non-zero entries in matrix $A$ is bounded by $2N^2$, where $N$ is the size of the $N \times N$ DEM. The upper bound of $2N^2$ is determined since there will be $N^2$ non-zero entries to load the identity matrix. All other non-zero entries represent flow from one cell to one other cell. There can be at most $N^2$ additional non-zero elements, since each cell can flow in only one direction. Taking advantage of the sparse nature of matrix $A$ the linear system can be solved efficiently. In Figure 2.4 we show the compute time to initialize and solve the linear system corresponding to the matrix size.

An example of solving the flow accumulation is as follows. Assume we have a trivial 3 by 3 elevation matrix, we the value at each index equals the height of the cell.

\[
\begin{pmatrix}
1 & 2 & 4 \\
3 & 9 & 5 \\
6 & 7 & 1
\end{pmatrix}
\]

Step 1 - Initialize the equations in the form $Ax = b$. Load the identity in matrix $A$ and assume each cell receives 1 unit of rainfall. $X_{1-9}$ equals the amount of flow at each cell.
Step 2 - Determine the direction of flow. Each cell flows to the lowest adjacent neighbor, for simplicity in this example only the 4 orthographic neighbors are considered.

\[
\begin{pmatrix}
1 & \leftarrow & 2 & \leftarrow & 4 \\
\uparrow & \uparrow \\
3 & 9 & 5 \\
\uparrow & \downarrow \\
6 & 7 & \rightarrow 1
\end{pmatrix}
\]

Step 3 - Add flow information into the equations. For example cell #4, flows all its water into cell #1. So we add 1 in row 1 column 4 in matrix \( A \). Each flow direction (arrow) above creates an additional non-zero entry into matrix \( A \). When accounting for all the flow directions, the linear system will be the following.

\[
\begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1
\end{pmatrix}
= 
\begin{pmatrix}
X_1 \\
X_2 \\
X_3 \\
X_4 \\
X_5 \\
X_6 \\
X_7 \\
X_8 \\
X_9
\end{pmatrix}
\]
Step 4 - Solve the equations. This will provide a matrix that contains how much water enters each node called the flow direction grid.

\[
\begin{pmatrix}
X_1 & X_2 & X_3 \\
X_4 & X_5 & X_6 \\
X_7 & X_8 & X_9
\end{pmatrix} = \begin{pmatrix}
6 & 3 & 1 \\
2 & 1 & 1 \\
1 & 1 & 3
\end{pmatrix}
\]

The major benefits of this approach is simplicity, scalability and it is consistent (there is never a flow loop). However a significant disadvantage is that this method does not account for flow sampling and dataset inaccuracies that often unrealistically block flow. A method for processing terrain to better handle these terrain errors is described in chapter 5 entitled Hydrology Conditioned Datasets.

### 2.2.1 Assigning Flow Directions to Plateaus using Connected Components

One additional computation step needs to be performed to deal with an important problem, which is the occurrence of plateaus. These are defined as regions where the flow direction can not be determined based on steepest decent flow.
To deal with these cases, the plateaus are first identified using a very fast variant of the Union-Find algorithm developed by Franklin and Landis [8]. The input is a $3N - 2$ by $3N - 2$ binary matrix and the output contains a list of components, with each component representing one plateau.

Figure 2.5 shows the plateaus and initial drainage network. Once identifying the flat areas, the cell directions are set using a similar strategy to Terraflow [15], where a breadth-first search assigns directions towards the root or spill point. Spill points are identified as cells in a flat component that contain a nonzero direction. In other words, a cell in the component must have at least one adjacent cell with a smaller elevation. Flat areas that have no spill points are determined to be sinks. The directions of every cell in a sink are assigned to flow to a single point.

After assigning directions to every plateau and sink, the final flow stage can be computed. The linear system of equations is modified to include the directions assigned to the plateaus and sinks. The flow is recomputed and the final flow accumulation grid and flow direction matrix is determined. Figure 2.5 shows our final flow computation with the drainage network and watersheds.

![Figure 2.5: (a) Drainage network (white) before handling flat regions. (b) Drainage network (black) and watershed boundary (white) after accounting for flat regions.](image)

Virtually the exact same approach for finding plateaus can be used to determining watersheds. The only difference is that the flow direction matrix is used to initialize the connected components input, instead of the elevation matrix. For each flow direction, we set the connecting binary number from a 1 to 0, exactly the same as we would do for two adjacent elevations of the same height when finding
the plateaus above. In figure 2.6 is a visualization of an isolated watershed found using the connected components program.

Figure 2.6: The connected components approach can also be used to determine watersheds. This image shows an isolated watershed with the boundary shown in black.

To better illustrate how connected components is useful in determining watersheds and plateaus, a simple example will be shown below. Using the same initial DEM used to solve for the flow accumulation matrix, we will now solve for the watersheds.

First we initial a $2N - 1$ by $2N - 1$, where $N$ is the size of the $N \times N$ DEM. Also note that for simplicity flow directions are limited to 4 possible orthographic directions. In practice the D8 model is used and flow can travel in diagonal directions and this will also force our connected component matrix to be larger.

$$
\begin{pmatrix}
0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0
\end{pmatrix}
$$

Notice that there are nine 0 entries in the matrix above. Each of those zero values directly maps to the elevations in the DEM. Below we see the DEM along with the flow directions.
For each flow direction above, we set one cell from one to zero.

\[
\begin{pmatrix}
1 & 2 & 4 \\
\uparrow & \uparrow \\
3 & 9 & 5 \\
\uparrow & \downarrow \\
6 & 7 & 1
\end{pmatrix}
\]

Using this as input into the connected components program, the number and location of watersheds is determined. For this simple example there are only 2 watersheds. Hence, the connected components implementation will output 2 components and their locations.

Watersheds can be used to verifying the effectiveness of a terrain simplification algorithm and also for segmentation and environmental analysis.
3. Hydrology-Aware Terrain Simplification

3.1 Terrain Simplification to Preserve the Hydrology

In Figure 3.1, we show a flow chart describing the ridge-river terrain simplification technique for compressing and uncompressing the hydrology structure of a terrain. Inputs are shown in boxes and programs are shown in circles. The method is based on ODETLAP which reconstructs an approximation of a terrain using a set of points. The recovered terrain achieves a more accurate representation as more points are included.

The basic idea the ODETLAP point selection is to include the most important points that lie on the river and ridge networks. The Drainage Network Program implements the drainage network computation described before and outputs a set of points composing of ridges and rivers. Since these points are clustered together they add little value to a point selection compression technique, thus the Douglas-Peucker algorithm is used to reduce the number of points required to represent each river/ridge segment. This line simplification uses a error tolerance that defines the maximum a simplified line can deviate from the original. These refined pointes are used to represent the terrain.

Figure 3.1: Flow chart of the ridge-river technique. Inputs are in boxes and programs in circles.
3.1.1 Approximating Terrain using Over-determined Laplacian PDEs

To reconstruct the terrain from a subset of the original elevation data, we use Over-determined Laplacian Differential Equations (ODETLAP). The input to this method is a compressed subset of points and the output is the reconstructed surface geometry. The Laplacian PDE is extended by adding a new equation to form an over-determined system so that we can control the relative importance of smoothness versus accuracy in the reconstruction. Benefits of the method include the ability to process isolated, scattered elevation points and the fact that reconstructed surface could generate local maxima, which is not possible in the original Laplacian PDE by the maximum principle.

ODETLAP can process not only continuous contour lines but isolated points as well. The surface produced tends to be smooth while preserving high accuracy to the known points. Local maxima are also well preserved. Alternative methods generally sub-sample contours due to limited processing capacity, or ignore isolated points.

Since we are working on single value terrestrial elevation matrix, we have the Laplacian equation for every unknown non-border point.

\[ 4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} \]  

(3.1)

In terrain modeling this equation has the following limitations:

- The solution of Laplace’s equation never has a relative maximum or minimum in the interior of the solution domain, this is called the maximum principle, so local maxima are never generated.

- The generated surface may droop if a set of nested contours is interpolated

To avoid these limitations, an over-determined version of the Laplacian equation is defined as follows: apply the equation (2) to every non-border point, both known and unknown, and a new equation is added for a set \( S \) of known points:

\[ z_{ij} = h_{ij} \]  

(3.2)
Where $h_{ij}$ stands for the known elevations of points in $S$ and $z_{ij}$ is the computed elevation for every point, like in equation (2). The system of linear equations is over-determined, i.e., the number of equations exceeds the number of unknown variables, so instead of solving it for an exact solution (which is now impossible), an approximated solution is obtained by setting up a smoothness parameter $R$ that determines the relative importance of accuracy versus smoothness.

3.1.2 ODETLAP Point Selection

In prior work [18], determining which points to input into ODETLAP was based on certain geometric algorithm including Triangulate Irregular Network, Visibility test, Level Set Component that discovers important points which reflect the terrain structure and use our extended Laplacian PDE to approximate the terrain from these points. In this paper the goal is not only to preserve overall terrain structure, but also to ensure that hydrology important features are preserved as well. Our experiments have shown that points on the ridge network and drainage network are the most effective in capturing the hydrology. The ridge-river technique computes both the rivers and ridges, and then simplifies the line network to capture the most significant points.

The drainage and ridge networks are simplified using the Douglas-Peucker[6] line refinement algorithm. This algorithm selects the most significant points need to reconstruct a line within a given error tolerance. This tolerance specifies the maximum distance the line can deviate from the original. Therefore the higher the tolerance, the few points required and the greater the difference between the original network and the reconstructed network. The output from the Douglas-Peucker algorithm is an ordered list of the most significant points needed to reconstruct the line. These points represent our compressed version of the hydrology. As Figure 3.2 illustrates, when the tolerance is set appropriately there is a significant reduction in number of points with the difference in the reconstructed lines being negligible.

To uncompress, we first connect the ordered set of Douglas-Peucker elevation points by using the Bresenham[2] line rasterization algorithm. These points are used as input into ODETLAP which is used to “fill in” the missing data points and is
Figure 3.2: Simplifying the original drainage network using Douglas-Peucker. The refined line network is reduced by a factor of 3 with little visible difference.

Our compressed version of the hydrology exists as a subset of elevations from the original DEM, plus the points along the reconstructed line using the Bresenham line rasterization algorithm. All the points are selected along drainage significant features. The reasoning is that we want to capture the most significant points that preserve the hydrology. Therefore, the known points are incorporated into equations 3.3 and 3.4 depending on whether the point came from the river or ridge network respectively. Figure 3.3 shows the ridge-river network.

3.1.3 Recovering Terrain Using a ODETLAP Hydrology Customization

To more accurately capture the structure of the hydrology, the ODETLAP equations are modified for points selected on the ridge-river network. This drastically reduces the amount of error introduced, as shown in Figure 3.3. The modification assume that points on the drainage network are slightly smaller than the average of their 4 neighbors so for river points we can modify equation 3.1 as follows:

$$4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} - D_R$$

(3.3)

where $D_r$ stands for decrement for the rivers, this variable is an integer corresponding the number of meters the rivers lie below the average of the 4 neighbors. Similarly,
ridge network points are higher than the average of their four neighbors, thus for selected ridge network points, the equation becomes:

$$4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} + I_R$$

where $I_R$ is an integer corresponding to the increment for the ridges. Experimentation has shown that setting $D_R = I_R = 2$ has been effective. In future work we plan to study how varying this parameter affects the results and investigate ways to automatically select an optimal value.

Figure 3.3: Modifying the ODETLAP equations to better represent ridges and rivers has a drastic decrease in the amount of hydrology error. Both plotted lines above use the same set of points.

3.2 Effectiveness of Ridge-Ridge ODETLAP Simplification

The primary focus of this paper has been to describe a metric that accurately captures the amount of error introduced into a reconstructed drainage network. Using this metric, we have been developing an algorithm for achieving high compression ratios without significantly altering the hydrology. The current effectiveness of this approach is shown in the table below. A very common terrain compression is JPEG2000 [3]. This method obtains a low percentage of cells that flow uphill.
This correlates to a fairly low hydrology error. The ridge-river technique described in this paper is effective in achieving high compression ratios with a fairly low error; however, it currently does not consistently beat JPEG2000. We strongly believe that small modifications to the current ridge-river method will allow us to achieve a significantly better hydrology error. We are investigating further modifications to the ODETLAP equations, and to automatically select optimal parameters.

<table>
<thead>
<tr>
<th>DATA</th>
<th>Ratio</th>
<th>Ridge-River % Uphill</th>
<th>Ridge-River ERROR</th>
<th>JPEG2000 % Uphill</th>
<th>JPEG2000 ERROR</th>
<th>TIN % Uphill</th>
<th>TIN ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>hill1</td>
<td>13</td>
<td>2.05%</td>
<td>0.0023</td>
<td>0.12%</td>
<td>0.0020</td>
<td>0.79%</td>
<td>0.0432</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>3.16%</td>
<td>0.1149</td>
<td>0.18%</td>
<td>0.0030</td>
<td>1.11%</td>
<td>0.0502</td>
</tr>
<tr>
<td></td>
<td>54</td>
<td>2.46%</td>
<td>0.2316</td>
<td>0.24%</td>
<td>0.0082</td>
<td>1.33%</td>
<td>0.0600</td>
</tr>
<tr>
<td>hill2</td>
<td>14</td>
<td>0.85%</td>
<td>0.0005</td>
<td>0.21%</td>
<td>0.0010</td>
<td>1.25%</td>
<td>0.0333</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>1.21%</td>
<td>0.0063</td>
<td>0.31%</td>
<td>0.0017</td>
<td>1.80%</td>
<td>0.0304</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>1.39%</td>
<td>0.0129</td>
<td>0.46%</td>
<td>0.0047</td>
<td>2.43%</td>
<td>0.0421</td>
</tr>
<tr>
<td>hill3</td>
<td>11</td>
<td>2.65%</td>
<td>0.0026</td>
<td>0.10%</td>
<td>0.0059</td>
<td>0.76%</td>
<td>0.0311</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>4.33%</td>
<td>0.0075</td>
<td>0.11%</td>
<td>0.0051</td>
<td>0.77%</td>
<td>0.0343</td>
</tr>
<tr>
<td></td>
<td>47</td>
<td>2.70%</td>
<td>0.0100</td>
<td>0.13%</td>
<td>0.0161</td>
<td>0.85%</td>
<td>0.0405</td>
</tr>
<tr>
<td>mtn1</td>
<td>16</td>
<td>3.75%</td>
<td>0.0267</td>
<td>0.41%</td>
<td>0.0026</td>
<td>3.96%</td>
<td>0.0563</td>
</tr>
<tr>
<td></td>
<td>39</td>
<td>4.96%</td>
<td>0.0530</td>
<td>0.80%</td>
<td>0.0036</td>
<td>5.11%</td>
<td>0.0583</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>5.91%</td>
<td>0.0611</td>
<td>1.33%</td>
<td>0.0067</td>
<td>6.28%</td>
<td>0.0667</td>
</tr>
<tr>
<td>mtn2</td>
<td>16</td>
<td>3.93%</td>
<td>0.0769</td>
<td>0.40%</td>
<td>0.0033</td>
<td>4.42%</td>
<td>0.0748</td>
</tr>
<tr>
<td></td>
<td>38</td>
<td>5.15%</td>
<td>0.1169</td>
<td>0.75%</td>
<td>0.0033</td>
<td>5.72%</td>
<td>0.0874</td>
</tr>
<tr>
<td></td>
<td>59</td>
<td>6.21%</td>
<td>0.1377</td>
<td>1.32%</td>
<td>0.0067</td>
<td>7.09%</td>
<td>0.0904</td>
</tr>
<tr>
<td>mtn3</td>
<td>15</td>
<td>3.10%</td>
<td>0.0254</td>
<td>0.40%</td>
<td>0.0015</td>
<td>4.16%</td>
<td>0.0592</td>
</tr>
<tr>
<td></td>
<td>39</td>
<td>4.33%</td>
<td>0.0493</td>
<td>0.78%</td>
<td>0.0027</td>
<td>5.63%</td>
<td>0.0624</td>
</tr>
<tr>
<td></td>
<td>61</td>
<td>5.13%</td>
<td>0.0639</td>
<td>1.40%</td>
<td>0.0050</td>
<td>6.63%</td>
<td>0.0650</td>
</tr>
</tbody>
</table>

Table 3.1: The amount of potential energy error for six 400 by 400 datasets sampled at 30m resolution. The percentage of flow traveling uphill is also shown, along with the compression ratio of each dataset.

Visual inspection of the reconstructed drainage networks correspond to the measurement determined by the potential energy metric. This is observed in Figure 3.4, where the higher error correlates to fragmented and uphill drainage networks. The modular design of our terrain simplification approach allows substituting different algorithms in place of the ones focused on in this paper. For instance, Terraflow or ArcGIS could be used to compute the ridge-river network. Also, a different line simplification technique could be used instead of Douglas-Peucker. This
Figure 3.4: The images show the a 400 × 400 hill2 dataset sampled at 30m resolution and compressed using the ridge-river technique. The color regions represent the elevations with blue being low and red corresponding to high elevation. The black region shows the significant drainage network above the threshold of 100. The higher potential energy error metric correlates with a visible difference in the drainage network. Notice how the high error corresponds to short fragmented drainage networks.

allows modification to fit the specific objectives of the user and application.

Points on the ridges of the terrain, as well as the rivers are important for preserving the hydrology. Rather then use an existing algorithm, we discovered that inverting the terrain and running the drainage network provides a quick, effective method for approximating the ridge network. Once again, this approach can be done with any drainage network program. The ridges are important in terrain compression for extracting and exploiting terrain structure, but also have other GIS applications such as visibility siting, hydrology and edge detection.
4. Hydrology-Aware Triangulation of Terrain

We present a new data structure called HydroTIN for simplifying terrain that captures hydrology significant features using a hydrology-aware Delaunay triangulation. This triangulation preserves the hydrology by using irregular-sized, non-overlapping planes to model regions that flow in a uniform direction. Edges are associated with drainage and ridge networks that incorporate physically-based structure into the model without significant overhead. This allows better compression ratios the standard Triangulated Irregular Networks with higher hydrology accuracy. Standard error metrics such as root mean squared (RMS) and maximum error fail to capture whether a reconstructed terrain accurately captures the hydrology. A hydrology error metric is used to verify our results based on the potential energy required for the reconstructed drainage to flow on the original terrain.

Figure 4.1: HydroTIN is a targeted simplification technique that incorporates the structure of the ridge and rivers and optimized for hydrology preservation.

4.1 Prior GIS Triangulations

Triangulated Irregular Networks or (TIN) [12] is a very popular algorithm for storing a surface for GIS applications. With TIN the surface is stored as a network
of non-overlapping, irregular sized and oriented triangles. This is different then
a Digital Elevation Model (DEM) which is a dense raster elevation matrix with
the \(x\) and \(y\) index containing the elevation of \(z\). TIN selects a subset of the \(n\)
most significant DEM points, these points are then used to construct a Delaunay
Triangulation. The Delaunay triangulation is a common computational geometry
algorithm that maximizes the minimum angle of all the triangles in the mesh. In
our implementation we use the divide and conquer Delaunay triangulation [9] which
runs in \(O(n \log n)\) time.

Selecting the appropriate points to use for the Delaunay Triangulation is cru-
cial in constructing an accurate representation of the surface geometry without
significantly distorting terrain important features. The selection process begins by
defining the boundary region of the terrain. This is constructed by dividing terrain
into two large triangles that encompasses the entire terrain region. To minimize
the maximum error, a point is added one at a time. The point that is the farthest
distance from the terrain reconstruction is inserted into the TIN. The terrain is then
reconstructed and the average and maximum error is recomputed. This process of
inserting more points occurs one at a time until some error threshold is achieved. In
this paper, the notion of ‘distance’ is redefined. Instead of distance being equal to
the absolute value of elevation between the TIN and the original DEM, distance is
defined using a hydrology error metric. Therefore, instead of minimizing maximum
or RMS (root mean squared) error, our HydroTIN algorithm minimizes the amount
of drainage network error based on a potential energy error metric. Before selecting
points based on their “hydrology distance”, the triangle mesh is initial seeded using
significant points on the drainage network and ridge network of the terrain.

Numerous methods have been developed for estimating a drainage network
from a specified segment of terrain. Based on the fact that elevation data is only an
approximation for the actual terrain, some methods allow for water to flow uphill
until spilling over an edge. These flooding methods determine spill points out of
every basin. In the Terraflow approach [5, 15], the path of least energy is used to
flow uphill until reaching the spill point. The flow runs uphill in situations when
there is not an adjacent lower elevation. These methods often keep expanding the
drainage networks until they flow off the edge of the terrain. This is because it is assumed that the initial input DEM is prone to collection and sampling errors that cause unrealistic depressions.

Past work has been done for defining a metric for comparing how well a computed drainage compares to the real world drainage [17]. The D8 model can assign flow in one of the eight possible directions. In the SFD (single flow direction) version of the D8 model the entire amount of flow from each cell is entirely distributed to the lowest adjacent neighbor. This is not the case in the MFD (multi-flow direction) version the flow is fractionally distributed to all the lower adjacent neighbors.

A slightly more sophisticated MFD model is the $D_\infty$ model. As the name indicates, flow can travel in an infinite number of directions and is not limited to eight directions. The amount of water leaving each cell is distributed to one or more adjacent cells based on the steepest downward gradient [14, 13].

To simplify each curve in the ridge-river network, Douglas-Peucker [6] is used where the reconstructed curve can not deviate more than a predefined tolerance from the original. The output from Douglas-Peucker is a subset of the original points with the first and last point on the curve always existing in the refinement. The number of points used to reconstruct the curve is inversely related to the error tolerance.

### 4.2 HydroTIN Algorithm

A Delaunay triangulation requires a set of points that will become the vertices of the triangle mesh. Therefore the algorithm for computing HydroTIN selects points considered optimal for preserving the hydrology information. Past work has shown that points on the River and River network are important for hydrology preservation. This Ridge-River network is simplified using the Douglas-Peucker line refinement algorithm to select the most significant points along each ridge-river segment. Figure 3.2 shows this simplification can significantly reduce the number of points required with the difference between the reconstructed line and the refined line being negligible. The refined Ridge-River points are the initial vertex seeding of the triangulation.

Once the initial triangulation is determined, a special case needs to be ad-
dressed. Ideally, there will be no triangles that are formed that have all three vertices lie on the river network. These triangles cause problems since they flatten out valleys crucial in preserving drainage information. Depending on the input parameters and the terrain dataset, in practice roughly $\frac{1}{5}$ of the triangles will have all 3 vertices on the river network. To correct this problem, the point with the highest elevation within each of these triangles is inserted into the triangulation. Triangles can still form such that they do not contain at least one ridge points or one of the newly insert high elevation points. However, the vast majority are corrected once the Delaunay triangulation is recomputed. Figure 4.2 shows the triangulation after the initial drainage structure has been incorporated.

At this stage the flow is computed on the reconstructed terrain in order to identify locations at risk for high error. Once identified, the $k$ most points with the highest error are added to the triangulation. The flow is then computed and the process iterates until the amount of error falls below a certain predefined threshold.
Figure 4.3: Left: Shows ridge-river points refined using Douglas-Peucker. This is the initial "seeding" of the triangulation. Right: Triangulation derived from the nodes on the left. Notice how the edges align with the ridge and river networks.

The optimal value of $k$ is 1, however since the triangulation only effects a local area of the terrain simplification, adding more points a certain distance away from the other points will not effect the reconstruction. Adding more then one point, one can encounter a problem if the points are too close together since the refined points are sometimes clustered. This is because real terrains are mostly continuous so if one point is far away, adjacent points are also likely to be erroneous, and will be selected as well. Because of this, refined points selected by any of our strategies may be redundant in some regions, which is a waste of storage.

Figure 4.4: Forbidden zone
We perform a check process when adding new refined points: the local neighbor of the new point is checked to see if there is any existing refined points which were added in the same iteration. If yes, this new refined point is discarded and point with the next biggest error is tested until we find desired number of refined points. So as shown in figure 4.4, all potential refined points that are close to an existing refined point (green points) are useless (marked red), and only points that are beyond some distance from green points are selected (marked yellow). By using this method we can add more points per iteration, drastically reducing compute time.

4.3 Results

![Energy Ratio Metric - graph](image)

Figure 4.5: Amount of potential energy error determined when mapping the flow direction matrix from the reconstructed terrain onto the original elevation matrix. Error is weighted by the gradient and the amount of flow.

To verify the effectiveness of HydroTIN, our results are compared against terrain computed using a Triangulated Irregular Network (TIN). The specific TIN implementation used in our comparison selects a point that is the farthest away from the reconstructed triangle mesh. Only one point is selected at each iteration.

Figure 4.5 shows an error plot of HydroTIN versus TIN computed using a
400 × 400 DEM sampled at 30 × 30 meter resolution of the Hawaiian island of Oahu. For HydroTIN, the initial seeding using the refined ridge-river points contains a relatively high potential energy error. This first stage is important in approximating the core drainage structure, however the error is high due to localized areas of the triangle mesh that have inaccurate slope and contains regions where the flow is traveling uphill when compared to the original. After the first iteration when the 100 points with the highest error are incorporated into the triangulation, the error drops dramatically.

Figure 4.6: Percentage of cells that are flowing uphill when mapping the flow direction matrix obtained from the terrain reconstruction onto the original elevation matrix.

After a couple of iterations HydroTIN will consistently achieve a lower error then TIN, roughly by a factor or 10. Both approaches tend to converge around a certain error value. For TIN, this convergence is nearly immediate since adding new points based on maximum error tends to result in short, fragmented drainage networks that are not representative of naturally occurring flow. These fragmented rivers tend to have a relatively small number of cells contributing in the watershed.

Not surprisingly, there is a high correlation between the error and the number of points that are flowing uphill. In figure 4.6 the relationship between the number
of points and the percentage of points flowing uphill is shown. As more points are included the numbers of cells flowing uphill are reduced for both implementations. However, only a very small decrease is observed for TIN. In contrast, HydroTIN consistently drops the number of upward flowing points. In the last iteration, the number of cells flowing up hill for HydroTIN is a mere 319 out of 160,000, or 0.2%. For TIN, 2,864 cells flow uphill or 1.79%.

A user defined input determines the tradeoff between compression size and the amount of potential energy error. The program will continue to iterate until the error is below a user defined threshold. Figure 4.1 is a visualization of HydroTIN which is stored by compressing less then 3500 points and achieves an error of 0.01. The edges of the triangles tend to follow the ridge and river networks preserves drainage structure and preventing significant blocking of water flow. After two iterations of minimizing the amount of potential energy error drops by a factor of 10 by adding just 200 points.
5. Hydrolgy-Conditioned Datasets

5.1 Introduction

Drainage network algorithms often provide fragmented river segments because of small sinks in the approximated terrain. Depressions of a tiny size such as only one cell can be fixed using a simple median filter. However, the challenge of fixing much larger depressions is a difficult problem to overcome. Approaches such as Terraflow [15] will find the path of lowest energy uphill out of these depressions. This yields long continuous river segments. However, this does not accurately model the physical properties of water flow. This section describes a method for modifying the initial Digital Elevation Model (DEM) such that the resulting drainage network will have small insignificant ridges removed from the DEM (Figure 5.2). This allows water to flow passed these areas using a standard physically-based drainage algorithm such as steepest-decent flow.

5.2 Results

Our reconstructed terrain captures the important aspects of the drainage network while still achieving a high compression rate. The reconstruction also has a more natural and realistic representation of the original hydrology because small insignificant ridges have been removed in the point selection process. This results in larger, fewer watersheds. The recomputed drainage network is also captured accurately, besides the small tributaries which aren’t considered of high importance. We
Figure 5.2: LEFT: Original Terrain with watersheds (white) and drainage network (black). The flow here is blocked by a small ridge that occurs due to sampling and data collection errors. RIGHT: Flow passes through small insignificant barriers. Allowing larger and more realistic watersheds and drainage networks can also store the compressed version using far fewer points than the original DEM. The user can define the level of detail and hence the number of points by adjusting the tolerance level for the Douglas-Peucker algorithm.

Figure 5.3 shows the drainage network computed on four instances of a 400x400 elevation matrix representing a segment of the Hawaiian island of Oahu. In (a) the hydrology was computed on the original elevation matrix. (b) and (c) correspond to hydrology computed on the reconstructed terrain using ODETLAP, where in (b) the points were selected using our ridge-river technique described above, and in (c) using the original ODETLAP method (in each iteration the $k$ “farthest points” were included). In (d) the hydrology is computed on a terrain is recovered from a lossy JPEG2000 compression. All the reconstructions have a similar RMS error of about 8.5.

The first results showed that the hydrology consistency is better preserved on terrain recovered based on the ridge-river point selection method than using original ODETLAP point selection and JPEG2000. Another interesting investigation would be to use the drainage network as a model of natural terrain formation. This could be used to extract structure from the terrain for segmentation and division, allowing for better compression.
Figure 5.3: The watersheds (white) and drainage networks (black) on the original and recovered terrains.
CONCLUSION

The potential energy metric introduced in this paper provides a quantitative measurement of the amount of error introduced by a terrain simplification technique. This value is reflective of the visible examination of the drainage networks, with higher error corresponding to fragmented and unrealistic flow directions (flow traveling uphill).

The original DEM is an approximation of the real world terrain surface and not the complete truth, due to dataset and sampling errors. Therefore it would be inaccurate to compare one drainage network computed on the original DEM versus the drainage network computed on the reconstructed surface. Instead, the drainage network is computed on the reconstructed surface and compared against the original terrain. Flow can travel in different directions then the original drainage network, yet contain a low error metric if the flow directions are reasonable. Standard error metrics such as standard metrics such as root mean squared error and maximum error are ineffective in evaluating the amount of error introduced, as they do not take into account important hydrology features.

As more terrain is being sampled at ever increasing resolutions, it becomes more important to be able store and manipulate large elevation datasets and evaluate the amount of error introduced by lossy compression. However, current techniques for compressing these datasets lose important information that is essential for running operations on the reconstructed geometry with reliable results. Understanding how compression affects important terrain structure, such as hydrology, allows the GIS community to understand how a compression technique affects the drainage accuracy of the reconstructed terrain. Our targeted compression technique has the goal of minimizing the amount of potential energy error, thus allowing for high compression ratios with minimal loss of hydrology information, while at the expense of other terrain structure. The net result of this work is a compression scheme and error evaluation metric with applications including flooding, erosion, sanitation, and environmental protection.
REFERENCES


REPRESENTATION, COMPRESSION AND PROGRESSIVE TRANSMISSION OF DIGITAL TERRAIN DATA USING OVER-DETERMINED LAPLACIAN PARTIAL DIFFERENTIAL EQUATIONS

By

Zhongyi Xie

A Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Approved:

W. Randolph Franklin, Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

March 2008
(For Graduation May 2008)
LIST OF TABLES

3.1 Impact of forbidden zone: Size of forbidden zone = 5. Note how the errors are much smaller when a forbidden zone is used . . . . . . . . . . 14

6.1 ODETLAP Compression results: Compress 3 hilly and 3 mountainous data sets using ODETLAP algorithm and further compression mentioned in section 4.1. Root-Mean-Square elevation and slope errors are recorded . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

6.2 Progressive Transmission: Compressed size of points sent are shown, with the corresponding error given as the column title. Algorithm ends when RMS error of the reconstructed surface < 10. Compressed results of our method and Bzip2 are compared, RLE+LP means Runlength encoding for \((x, y)\) and linear prediction for \(z\). . . . . . . . . . . . . . . 23
LIST OF FIGURES

3.1 Triangulated Irregular Network .................................. 6
3.2 Impact of $R$ in ODETLAP interpolation [1]: (a): The Square Contours to be Interpolated. (b): Lagrangian Interpolation. (c): Overdetermined Solution, $R = 1$. (d): Overdetermined Solution, $R = 10$. .......... 9
3.3 Algorithm Outline .................................................. 9
3.4 Visibility Index: Points are selected according to their visibility indices. 12
3.5 Forbidden Zone: Points are often clustered as in the left figure, hence the reconstructed surface is not very accurate; We apply a forbidden zone in the refined points selection so that points are no longer clustered as in the right figure. Reconstructed surface is more accurate as we can see in table 3.1. ................................................................. 13
3.6 Forbidden Zone: Points too close are ignored ..................... 14
4.1 Histogram of two DEMs, we can see most runs are below 512. ...... 16
4.2 Compressing $z$ values using linear prediction. ...................... 17
5.1 Progressive transmission flowchart.eps ................................ 19
5.2 Flow chart of the ridge-river technique. Inputs are in boxes and programs are in circles.[2] ......................................................... 20
6.1 Test datasets: 400 by 400 resolution .................................. 21
6.2 Progressive example: Hilly dataset(hill2): Compressed sizes are 1.2KB, 3.5KB and 7.1KB .............................. 23
6.3 Progressive example: Mountainous dataset(Mtn1): Compressed sizes are 1.3KB, 3.6KB and 7.3KB .......................... 24
6.4 Compressed size at different stages of compression for both our method and bzip2. We list 5 stages: when initial points are selected, when RMS error drops below 75%, 50%, 25% of the initial RMS error for the first time and when RMS error is smaller than 10. ............................. 25
6.5 Root-Mean-Square error of reconstructed surface is plotted against the size of compressed points. As before, we use run length encoding and linear prediction to do the compression. .............................. 25
ACKNOWLEDGMENT

My greatest thanks go to my advisor, Dr. William Randolph Franklin. I have an immense amount of gratitude for the knowledge, support and guidance that helped me through my graduate studies. I feel fortunate to have Prof. Franklin as my advisor, teacher and friend.

I want to express my gratitude to Dr. Barbara Cutler of RPI, Dr. Marcus Vin-cius Alvim Andrade of Universidade Federal de Viçosa, Viçosa, Brazil and Franklin Luk of RPI and Hong Kong Baptist University for helping me throughout my study at RPI. I also want to thank my fellow student Metin Inanc for his support and encouragement.

I would also like to thank the assistants in the computer science department for their help, kindness and caring, namely, Terry Hayden, Jacky Carley, Chris Coonrad, and the lab staff for all their help throughout the years in solving many problems.

Nobody has given me as much encouragement, support and unconditional love throughout my graduate study and all aspects of my life, as my parents, Fuying Gong and Nanqing Xie and my girl-friend Yin Liu. I am forever grateful for their patience and understanding and belief in me in whatever I choose to pursue.

Along the way of my studies I was fortunate to have gained true friends with who I shared fun and memorable times, Chao Chen, Gary(Gehua) Yang, Zujun Shentu, Shun Yao, Hengfang Zhu, Yu Sheng, Pang Shushu, Lingzhi Luo, Ting Xie, Jacob ”Juda” Becker, Jonathan Muckell, Jared Stookey, Daniel Tracy, Pin Lu and many more. They were around at good times and bad times. Their presence made my life very colorful and I sincerely hope that I have had a similar impact on theirs.
ABSTRACT

We present current research on terrain (i.e., elevation) representations and operations thereon. The work is motivated by the large amounts of high resolution data now available. The significance of our method mainly lies in its ability to highly compress terrain data while preserving important features. The backbone of the method is our ODETLAP representation which generalizes a Laplacian partial differential equation by using two inconsistent equations for each known point in the grid, as well as one equation for each unknown point. The surface is reconstructed from a carefully-chosen small set of known points. The terrain we are using is Digital Elevation Model (DEM) data, stored as a raster of elevations. We combine ODETLAP with other computational geometry techniques including Triangulated Irregular Network (TIN), Visibility Test, Level Set so that the small set of known points could be effective in the reconstruction process.

Based on the novel ODETLAP representation, we propose a new surface compression technique to lossily compress terrain data. Our approach first approximates the uncompressed terrain using ODETLAP. Then the approximation is refined with respect to the original terrain by selecting more known points. These two steps work alternately until we find an approximation that is good enough. We then use run length encoding and linear prediction to further compress the point set to achieve a higher compression ratio.

In addition to terrain compression, we also propose a progressive terrain data transmission method based on ODETLAP. Concretely, our technique is capable of reducing a hilly DEM dataset to 1% of its original binary size and a mountainous DEM dataset to 3% of its original size. The corresponding RMS elevation errors are all below 10 meters. We study the tradeoffs between compressed size and reconstructed surface accuracy and demonstrate that our method is effective in transmitting terrain data over slow links and lays a concrete foundation for further operations like rendering and editing.

vii
1. INTRODUCTION

1.1 Motivation

In the past a few years the geographical information science (GIS) community has seen an explosion in data volume and great improvement in the accuracy of data acquisition. High-resolution terrain elevation data at 3m is now widely available through the USGS’s web site, which makes possible more realistic rendering of terrain features by all kinds of visualization software. Such growth of terrain data is reflected in the growth in number of internet sites offering terrain data as freeware, for example, United States Geological Survey’s DEM data sets [3]. Also we see great growth in Mobile GIS services and GPS units are more and more affordable and widely used. Digital Elevation Models (DEMs) represent the continuous surface of earth using a regular grid of samples which record the height value. It is a potential tool for terrain analysis at varied spatial and temporal scales.

One common goal for general compression implementation is to achieve maximum compression while minimizing processing time. However some users are concerned about the overall speed of the algorithm, while other users need optimal compression and are less concerned about processing time due to the availability of supercomputers.

1.2 ODETLAP Overview

We use Over-determined Laplacian Partial Differential Equations (ODETLAP) to approximate and lossily compress terrains. We construct an over-determined system using triangulation, visibility tests, level set components and random selection; then use an over-determined PDE to solve for a smooth approximation. The initial approximation might be very rough, i.e., it may contain considerable elevation or slope errors. After that, we refine the approximation with respect to the original terrain by adding into the important points set those points with biggest elevation error or slope error, and then use ODETLAP again on the augmented representation to find a better approximation. These two steps are alternately applied.
until a predefined maximum error is reached. We also study the size versus accuracy tradeoff and plot the error curve.

ODETLAP can process not only continuous contour lines but isolated points as well. The surface produced tends to be smooth while preserving high accuracy for the known points. Local maxima are also well preserved. Alternative methods generally sub-sample contours due to limited processing capacity, or ignore isolated points.

Due to the popularity of mobile devices that support terrain transmission and operations, it is now more practical to transmit terrain data in a progressive manner. We extend our ODETLAP algorithm to be able to progressively transmit terrain data through network.

1.3 Thesis Outline

This paper is organized as follows: chapter 2 gives a brief summary of existing algorithms that do terrain compression and progressive transmission, chapter 3 talks about how ODETLAP works, chapter 4 presents terrain compression and chapter 5 presents applications of ODETLAP in progressive transmission and hydrology. Test and experiment results are given in chapter 6. Chapter 7 summarizes the whole thesis and future work is given in chapter 8.
2. RELATED WORKS

2.1 Terrain Compression

Simplification and compression of three dimensional terrain/surface data share
the same goal but take different routes: simplification reduces the complexity of the
terrain by reducing the number of vertices and faces in the mesh while compression
works to compactly represent the connectivity data of the terrain. We will briefly
discuss some classical work in both areas.

2.1.1 Progressive Meshes

**Progressive meshes** (PM) is a surface representation introduced by Hoppe
[4]. The simplification is based on a pair of reversible operations: edge collapse and
vertex split, the first of which works for simplification and the latter for reconstruc-
tion/refinement. A PM maintains a sequence of refinement/simplification records
so that a mesh of any precision can be obtained by incremental refinement.

2.1.2 Triangulated Irregular Network

The **Triangulated Irregular Network** (TIN), a piecewise linear triangular
spline, first implemented in cartography in 1973 [5] is an approximated, lossy rep-
resentation that has the major advantage that it is not tied either to a particular
coordinate system, or to a developable surface. We use an implementation which is
greedy: find the point with biggest error and use it to refine the triangulation. For
more information, please see section 3.1.

2.1.3 Mesh Compression

Taubin and Rossignac propose a mesh compression scheme which records the
connectivity information into a spanning tree [6]. Their method is capable of com-
pressing connectivity information to 2 bits per triangle. Normals, colors, and texture
coordinates, are compressed in a similar manner. One of the disadvantages of this
method is its large memory requirement due to the requirement of random access
to all vertices in decompression.

2.1.4 Hierarchical Triangulation

Hierarchical Triangulation (HT) is a hierarchical triangle-based model
for representing surfaces over sampled data proposed by De Floriani [7]. Similar
to TIN, HT is based on subdividing the surface into nested triangulations which is
then organized into a tree where each node stands for a triangulation except the
root. In order to describe the surface of each triangles, HT associates values with
each vertices. HT is capable of extracting the representation of terrain at variable
resolutions over the domain.

2.1.5 Visibility Preserving Terrain Simplification

Ben-Moshe proposes a terrain simplification technique based on preserving
inter-point visibility relationships [8]. This technique aims at preserving visibility
information, which, informally speaking, means points in the original terrain that
are visible to each other (and respectively, points not visible to each other) are still
visible to each other (respectively, not visible to each other) after the simplification.
This technique is designed to meet the need of finding good locations on the terrain
to place “observers” (antennas, guards, etc). It works by first computing the ridge
network (a collection of chains of edges of the terrain). This ridge network induces
a subdivision of the terrain into patches and each patch is independently simplified
using one of the standard terrain simplification methods.

2.2 Progressive Transmission of Terrain Data

This section presents a short review of existing methods for progressive trans-
mission of geospatial data over the world wide web. Basically, the methods can be
divided in two categories: raster data and vector data transmission.

In general, most methods developed for raster data transmission are based on
image compression techniques since they can provide good compression with no (or
low) loss of information. Some simple methods randomly select subsets of pixels
from the image being transmitted and incrementally complete the image by adding pixels [9, 10]. Other more elaborate strategies [11, 12, 13] use some hierarchical structure, such as quadtree, to partition the image and select more pixels from those parts containing more details. Thus, the image quality can be incrementally improved by transmitting/including points in those image parts.

Other more sophisticated methods, for example [14, 15, 16, 17, 18, 19] are based on advanced compression techniques such as wavelet decomposition, JPEG compression and JPEG2000. In general, these compression methods decompose the data in rectangular sub-blocks and each block is transformed independently. Furthermore, the image data is represented as a hierarchy of resolution features and its inverse at each level provides subsampled version of the original image. Thus, it is quite natural to apply this strategy for progressive transmission.

Generally, raster data transmission over the internet is used for visualization. Raster progressive transmission methods are very efficient since visual meaning can be extracted from images at very low resolution. However, in some applications, the objects need to be manipulated or processed to extract some additional information. In this case, a vector representation is used.

Often, in terrain modeling and surface reconstruction and rendering, vector data are represented using triangular meshes [20, 21, 22] that can be compressed by several methods which are either based on optimal point decimation, such as [23, 24, 25, 26] or exploit the combinatorial properties of the mesh [27, 28, 6]. Although these compression methods can be used for progressive transmission, an important bottleneck is the high storage space required for vector representation (even after compression).
3. REPRESENTING THE TERRAIN

Contemporary terrain representation techniques can be categorized into two kinds: array based and triangle based. One dimensional array and two dimensional matrix data are simple to operate on and easy to store.

3.1 Triangulated Irregular Network

Franklin’s Triangulated Irregular Network algorithm [5] is a triangle based terrain representation which builds a triangulated irregular network (TIN) using a greedy insertion method to approximate a surface. Starting with a matrix of elevations, it first splits the points’ bounding square (or rectangle) into two triangles along the diagonal and associates each triangle with the points inside it. In the next step, it searches within each triangle for the furthermost point from the triangle’s plane. The farthest point of all is used to split the triangle into three new triangles (or two new triangles if the farthest point happens to be on an edge of the triangle). It uses a breadth first search to avoid starvation: a triangle is never split if there exists an undivided triangle that was created before it. An issue with this approach is that in some steps the insertion of the farthest point may temporarily increase the error, but usually, after some additional insertions, the error will be reduced even more. So, the overall tendency is for the error to decrease when new points are added.
3.2 Representation by Approximation of an Overdetermined Laplacian PDE

3.2.1 Definition

As implied by the name, the Over-determined Laplacian Approximation (ODET-LAP) comes from Laplace’s equation, whose solution at any point \((x, y, z)\) is equal to the average of the solution values in its neighborhood. We have the equation

\[
4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1}
\]  

(3.1)

for every unknown non-border point, which is equivalent to saying the surface satisfies Laplacian PDE,

\[
\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = 0
\]

(3.2)

In terrain modeling this equation has the following limitations:

- The solution of Laplace’s equation never has a relative maximum or minimum in the interior of the solution domain, this is called the “maximum principle” [29]; so local maxima are never generated.
- The generated surface may droop if a set of nested contours is interpolated [30]

To avoid these limitations, an over-determined version of the Laplacian equation is defined as follows: first apply the equation (3.1) to every non-border point, both known and unknown, and then a new equation is added for a set \(S\) of known points:

\[
z_{ij} = h_{ij}
\]

(3.3)

where \(h_{ij}\) stands for the known elevations of points in \(S\) and \(z_{ij}\) is the “computed” elevation for every point, like in equation (3.1).

Note that the system of linear equations is over-determined, i.e., the number of equations exceeds the number of unknown variables. Since the system is very likely to be inconsistent, instead of solving it for an exact solution (which is now impossible), an approximate solution is obtained by trying to keep the error as small
as possible. Equation (3.1) is almost satisfied for each point, making it the average of its neighbors, which makes the generated surface smooth there. However, since we have known points where equation (3.3) is almost valid, they are not necessarily equal to the average of their neighbors, which probably makes the surface not smooth there. This is especially true when we have adjacent known points, like points that define contour lines. Therefore, for points with multiple equations we can choose the relative importance of accuracy versus smoothness by adding a smoothness parameter when solving the over-determined system [1]. In our implementation, equation (3.1) is weighted by $R$ relative to equation (3.3) which defines the known locations. So a very small $R$ will approximate a determined solution and the surface will be more accurate while a very large $R$ will produce a surface with no divergence, effectively ignoring the known points. Figure 3.2 shows how different values of $R$ will affect the generated surface [5]. Subfigure (a) shows the four nested square contour lines that we try to approximate. Subfigure (b) gives the Lagrangian interpolation and we can see the undesired lines that surface normal is not continuous. Subfigure (c) and (d) are generated by ODETLAP and with $R$ equal to 1 and 10. So in (c) where we made the accuracy as important as smoothness, the surface is quite accurate compared to in subfigure (d). However, the visible contours means it’s not as smooth as subfigure (d).

This over-determined system allows for processing of isolated, scattered elevation points as well as continuous contour lines and produces a smooth surface while the error is minimized. The generated surface has local maxima inside the innermost contour and shows little or no evidence of the contours. Instead of interpolation, approximation is a more suitable term for this method because the reconstructed surface is not guaranteed to go through the input data points.

ODETLAP can be used as a lossy compression technique since the original terrain can be approximated with some error using the set of points $S$ for equations (3.1) and (3.3).
Figure 3.2: Impact of $R$ in ODETLAP interpolation [1]: (a): The Square Contours to be Interpolated. (b): Lagrangian Interpolation. (c): Overdetermined Solution, $R = 1$. (d): Overdetermined Solution, $R = 10$.

Figure 3.3: Algorithm Outline

3.2.2 Algorithm Outline

The ODETLAP algorithm’s outline is shown in figure 3.3 and the pseudo code is given below. Starting with the original terrain elevation matrix there are two point selection phases: firstly, the initial point set $S$ is built by any of the methods described in section 3.3.1 and a first approximation is computed using the equations (3.1) and (3.3). Given the reconstructed surface, a stopping condition
based on an error measure is tested. In practice, we have used the root-mean-square (RMS) error as the stopping condition. If this condition is not satisfied, the second step is executed. In this step, $k \geq 1$ points from the original terrain are selected by method described in 3.3.2 and they are inserted in the existing point set $S$; this extended set is used by ODETLAP to compute a more refined approximation. As the algorithm proceeds, the total size of point set $S$ increases and the total error converges.

**input** OriginalTerrain: $T$

**output** PointSet: $S$

1. $S = \text{InitSelection}(T)$
2. $\text{ReconstructedSurface} = \text{ODETLAP}(S)$
3. while $\text{RMS}(\text{ReconstructedSurface} - T) > \text{Max_RMS}$
   
4. $S = S \cup \text{Refine}(T, \text{ReconstructedSurface})$
5. $\text{ReconstructedSurface} = \text{ODETLAP}(S)$
6. return $S$

This process can be easily used for progressive transmission such that the points selection is done in the server end, based on ODETLAP. These points are sent to the client and the terrain is reconstructed in the client using ODETLAP as well. Notice that the client needs to know the value of smoothness parameter $R$ used in the server end.

### 3.3 Point selection strategies

As we have seen in section 3.2.2, there are two stages where points are selected: the initial point selection stage and the refined point selection. We discuss each of them below.

#### 3.3.1 Initial Points Selection

##### 3.3.1.1 Random Selection

This strategy is the most intuitive and easiest to implement. The basic idea is select points randomly. Using a good random number generator, this strategy
ensures that most parts of the terrain contribute to the final reconstruction. This strategy is fast and robust.

3.3.1.2 Visibility Index

Let $p$ be an observer, The visibility index (VIX) [31] is defined to be the fraction of points that can be seen. This index is defined considering only a small region around $p$, which generally, is determined by a circle centered at $p$ with a radius of interest $r$.

There are several ways to compute the VIX values and we use the one proposed by Ray and Franklin [31]: for each terrain point $p$, randomly select $k$ sample points within the radius of interest. Then run a line of sight connecting $p$ to each random point to decide if the point is visible or not. The VIX of $p$ is given by the ratio between the number of visible random points and sample size. As a result, the VIX values are only approximation to the exact values and they are highly dependent on how many random points are chosen. In our tests, we used $r = 25$ and $k = 10$.

In this point selection strategy we assume that points with small VIX values are more important to define the terrain skeleton than points with big VIX. So the initial set is built containing points with small VIX values. However, our selection of points needs to reflect the overall VIX value distribution. This is done using a probability distribution that establishes how likely a point with a particular VIX value will be selected. This probability distribution is defined using the VIX value (small values mean bigger probabilities) weighted by the VIX values distribution. Thus, points whose VIX value occurs more frequently have their probability multiplied by a higher factor. Figure 3.4 shows the selected points from the terrain.

3.3.1.3 Level set components

Using an adaptation of level set ideas [32], we segment the terrain based on points’ elevation. That is, suppose that the elevation values range from $h_{\text{min}}$ to $h_{\text{max}}$ and given an integer $k$, the interval $[h_{\text{min}} \cdots h_{\text{max}}]$ is divided into “elevation slices” of equal size $k$ (the last slice can be smaller). Then, each terrain point $p = (x, y, z)$ is associated with the corresponding elevation slice that contains the height $z$; more precisely, to the slice $[z_i \cdots z_{i+1}]$ such that $z_i \leq z < z_{i+1}$. Next, each elevation slice
is partitioned into connected components which are computed using 8-connectivity, (i.e., the horizontal, vertical and diagonal neighbors are checked)\(^1\)

As in the previous strategy, this point selection criterion uses a probability distribution defined considering the elevation slices’ area (i.e., the total number of points in all the connected components in the elevation slice) weighted by an “elevation slice importance” assigned assuming that the most important slices are those in the extremities (lowest and highest) height - the importance decreases uniformly toward the slice with medium height.

### 3.3.2 Refined point selection - Greedy algorithm

After the initial point set is obtained, ODETLAP is used to reconstruct the elevation matrix. This matrix has high error with respect to the original terrain, mostly due to the limited size of the initial point set. As shown in figure 3.3, refined points selection is applied and a set of additional points is chosen and added to the existing points set \(S\) to form the augmented points set. The way we choose new points is greedy (similar to 3.1): we find a set of points with greatest absolute vertical error. The size of the set in our experiments is intentionally kept small (10% or smaller) so that for a given total number of points, more iterations could be used to reduce the error as much as possible. This is actually a trade-off between accuracy and computation time. The augmented set \(S’\) is then given to ODETLAP to reconstruct a more refined approximation. The newly obtained approximation is

\(^1\)Of course, the elevation slice point association and the connected component computation can be done simultaneously just adapting the connected component computation to check if a neighbor is in the same elevation slice.
again examined with respect to the original terrain against our stopping condition, which is either:

1. Relative RMS: Compute the root mean square (RMS) error of the approximation and check if its ratio against the RMS error of the first approximation is smaller than a predefined threshold.

or

2. Absolute RMS: Define a value for the maximum acceptable RMS error.

### 3.3.3 Forbidden Zone

![Forbidden Zone](image)

Figure 3.5: Forbidden Zone: Points are often clustered as in the left figure, hence the reconstructed surface is not very accurate; We apply a forbidden zone in the refined points selection so that points are no longer clustered as in the right figure. Reconstructed surface is more accurate as we can see in table 3.1.

Using the refined point selection described in section 3.3.2, one can encounter a problem: the refined points are sometimes clustered (left part in figure 3.5). This is because real terrain is mostly continuous so if one point is far away from the surface, adjacent points are also likely to be erroneous, and will be selected as well. Because of this, refined points selected by any of our strategies may be redundant in some regions, which is a waste of storage.

We perform a check process when adding new refined points: the local neighbor of the new point is checked to see if there is any existing refined points which were added in the same iteration. If yes, this new refined point is discarded and the
Figure 3.6: Forbidden Zone: Points too close are ignored

point with the next biggest error is tested until we find desired number of refined points. So as shown in figure 3.6, all potential refined points that are close to an existing refined point (green points) are useless (marked red), and only points that are beyond some distance from green points are selected (marked yellow). The effect of the forbidden zone can be seen in the right part in figure 3.5: no dense clusters of points are present and all points are distributed more evenly within the whole terrain.

<table>
<thead>
<tr>
<th>Data</th>
<th>Avg Error</th>
<th>Max Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w/F.Z.</td>
<td>w/o F.Z.</td>
</tr>
<tr>
<td>Hill1</td>
<td>3.16</td>
<td>7.35</td>
</tr>
<tr>
<td>Hill2</td>
<td>8.34</td>
<td>14.8</td>
</tr>
<tr>
<td>Hill3</td>
<td>1.54</td>
<td>3.00</td>
</tr>
</tbody>
</table>

Table 3.1: Impact of forbidden zone: Size of forbidden zone = 5. Note how the errors are much smaller when a forbidden zone is used.

In table 3.1, we show how the forbidden zone affects the average error and maximum error in our experiment. We implemented our algorithm with and without the forbidden zone and tested it with three data sets. In this test, we begin with 300 points and in each iteration 50 refined points are added. The results after 14 iterations justify the use of the forbidden zone in our algorithm.

RPI Geo* Final Report

Xie Masters Thesis
4. OPERATIONS ON THE TERRAIN

4.1 Terrain Compression

4.1.1 Compressing Points

To improve the compression ratio, the points are compressed using the following strategies: the \((x, y, z)\) coordinates are split into \((x, y)\) pairs and \(z\) alone. The former are compressed using an adapted run length encoding method, described below in section 4.1.2. The \(z\) sequence is compressed using linear prediction and bzip2. This splitting process gives a good compression ratio, as shown in 6.2, because the \((x, y)\) values are restricted to the terrain size, while the \(z\) values are more “arbitrary”.

4.1.2 Run Length Encoding

The coordinates \((x, y)\) are different from \(z\) because they distribute evenly within the matrix dimension’s values, for example from 1 through 400, while \(z\) values appear to be more randomly distributed. The run-length encoding is a simple lossless compression technique which, instead of storing the actual values in the sequence, stores the value and the count of sequence containing the same data value. A run is just a consecutive sequence that contains the same data value in each element. Since the \((x, y)\) values correspond to positions in a matrix, then we need to store only a binary bitmap where each location indicating whether the corresponding point exists in \(S\) or not. There is no need to store the original \((x, y)\) pairs. Thus, given a binary matrix of size \(N \times N\), the method is the following:

For each run length \(L\), test if

1. \(L < 254\), then use one byte for it
2. \(254 \leq L < 510\), then use FFFE as a marker byte and use a second byte for \(L - 254\)
3. \(510 \leq L < 766\), then use FFFF as a marker byte and use a second byte for \(L - 510\)
4. If \( L \geq 766 \), then use FFFFFFFF as a two byte marker and use next two bytes for \( L \).

We can see from the histogram of the run length (Figure 4.1) that most runs are below 512, that means for most runs we need only 1 byte to store it. Here we assume all runs are shorter than 65535, which is a reasonable value for terrains of 400 \( \times \) 400 resolution.

![Figure 4.1: Histogram of two DEMs, we can see most runs are below 512.](image)

### 4.1.3 Linear Prediction

Unlike \((x, y)\) coordinates, \(z\) contains more redundancy due to the inherent redundancy in the original terrain. Normally terrain data contains a high degree of correlation and that means we may predict the elevation value from its neighbors. The method of linear prediction has been very successful in image processing [33].

The sequence \(z\) that we are going to compress contains elevation information in the selected points by the ODETLAP algorithm in section 3.2.2. Because points appear in the order they were selected, we need to order them by their corresponding \(x, y\) coordinates so that during reconstruction process, the correlation can be re-established. Linear predictors attempt to identify the redundancy in adjacent points.

There are several different ways to do linear prediction, and within the JPEG standard, there are seven modes of prediction [34]. However, most of them use neighborhood information from two dimensions which in our case do not exist. As
a result, we are only using the simplest mode which predicts the next entry in the sequence by the previous one.

Figure 4.2: Compressing z values using linear prediction.

The whole procedure is shown in Figure 4.2. Given the original data sequence, the predicted sequence is computed by predicting each element by the previous one. Their difference is stored in a new correction sequence, which will be compressed by some data compression software like bzip2.
5. APPLICATIONS

5.1 Progressive Transmission of the Terrain

Conventional terrain/map related software that requires the availability of all data does not support any operation before the transmission process ends. Despite the development of networking technology and hardware, terrain data are still relatively large (a few Gigabytes) compared to the network speed (below 1 Megabytes per second for most users). As a consequence, the slow speed of conventional transmission methods makes them far from interactive. Progressive transmission, was proposed as a solution to the above problem. The idea is to successively send the terrain data starting with a coarse simplification, and on the user end, visualize or start some other operation. Although progressive transmission may extend the total transmission time due to the extra time needed for multiple operations, it has advantages over the traditional method in the following ways. Firstly, the client does not have to wait until all the data are received before any operation or viewing can be done, which increases the interactivity of the system. Secondly, the client can determine whether the current surface is accurate enough before all the data are sent and received. This saves the transmission time and resource need and increases the flexibility and efficiency of the system.

Our ODETLAP based progressive transmission system consists of a server which first selects a subset of original points and sends those points to a client that reconstructs the surface using ODETLAP. After that, the reconstructed terrain is evaluated. If necessary, the client would request more points from the server, which could result in a more accurate approximation of the terrain. Figure 5.1 shows the whole process. The user end can then do more operations based on the reconstructed surface. While other existing progressive transmission methods transmit all the data from the server to the client, our progressive transmission system has the advantage that only a subset of points need to be sent, which saves a lot of networking resource as well as transmission time. As long as some data points reach the client side, the whole terrain can be approximated by ODETLAP, so the user can see the terrain
as early as possible. When more points arrive at the client side, it can reconstruct the terrain more accurately from the extended points set, which gives user a more detailed view of the terrain.

5.2 Hydrology

ODETLAP can also be used in terrain simplification to preserve hydrology[2]. Figure 5.2 illustrates the ridge-river terrain simplification technique for compressing and uncompressing the hydrology structure of the terrain. The idea is to use drainage network program to find the ridge and rivers and the Douglas-Peucker algorithm to reduce the number of points, and finally, use ODETLAP to reconstruct the terrain. For more information, please refer to [2].
Figure 5.2: Flow chart of the ridge-river technique. Inputs are in boxes and programs are in circles.[2]
6. TESTS AND RESULTS

6.1 Datasets

We have tested our algorithm on a benchmark of six real DEMs shown in Figure 6.1. Each DEM is on a 400 × 400 grid, with spacing of 30 meters. The first three (Sub-figures (a), (b) and (c) in Figure 6.1) are hilly data sets while the last three are mountainous.

![Figure 6.1: Test datasets: 400 by 400 resolution](image)

6.2 ODETLAP Performance Analysis

Table 6.2 gives a summary of our ODETLAP algorithm’s compression performance on the data in Figure 6.1. The table shows some information about each terrain, including elevation range, size and standard deviation of the original terrain data as well as the test results. For each dataset, we run a series of tests with different numbers of starting points, ranging from 400 to 4000. When refinement is needed, 10 percent of the points are added in each of the 10 iterations. After that we...
find the minimum number of points needed to get the elevation Root-Mean-Square error below 10. We see that in order to control RMS error, we need more points when the standard deviation is relatively large (> 150).

<table>
<thead>
<tr>
<th>Elev range</th>
<th>Hill1</th>
<th>Hill2</th>
<th>Hill3</th>
<th>Mtn1</th>
<th>Mtn2</th>
<th>Mtn3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig size</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
</tr>
<tr>
<td>Stand. Dev. of Elev.</td>
<td>78.9</td>
<td>134.4</td>
<td>59.3</td>
<td>146.0</td>
<td>152.4</td>
<td>160.7</td>
</tr>
<tr>
<td>Compr. size</td>
<td>2984B</td>
<td>5358B</td>
<td>1739B</td>
<td>9744B</td>
<td>9670B</td>
<td>9895B</td>
</tr>
<tr>
<td>Compr. ratio</td>
<td>107:1</td>
<td>60:1</td>
<td>184:1</td>
<td>33:1</td>
<td>33:1</td>
<td>32:1</td>
</tr>
<tr>
<td># pts selected</td>
<td>1040</td>
<td>2080</td>
<td>520</td>
<td>4160</td>
<td>4160</td>
<td>4160</td>
</tr>
<tr>
<td>Elev. RMS error</td>
<td>8.49</td>
<td>9.93</td>
<td>8.31</td>
<td>9.48</td>
<td>9.55</td>
<td>9.68</td>
</tr>
<tr>
<td>Elev. RMS error perc.</td>
<td>1.68%</td>
<td>1.33%</td>
<td>1.66%</td>
<td>0.91%</td>
<td>1%</td>
<td>1.23%</td>
</tr>
<tr>
<td>Slope RMS error</td>
<td>2.81°</td>
<td>5°</td>
<td>1.65°</td>
<td>8.34°</td>
<td>8.36°</td>
<td>7.87°</td>
</tr>
</tbody>
</table>

Table 6.1: ODETLAP Compression results: Compress 3 hilly and 3 mountainous data sets using ODETLAP algorithm and further compression mentioned in section 4.1. Root-Mean-Square elevation and slope errors are recorded.

### 6.3 Progressive Transmission Results

The compressed size is given in table 6.2. We compress output \((x, y, z)\) triplets from ODETLAP using runlength for \((x, y)\) linear prediction for \(z\). We also compress the same points using bzip2 for comparison. The number of points is also listed. The six test data are rendered in Figure 6.1 for reference. They are all 400 by 400 DEM datasets. As we described in sections 4.1.3 and 4.1.2, the points are separately processing and compressed using run-length encoding and linear prediction. The results show that our compression method is effective and, in all circumstances it has a smaller size than bzip2.

We test progressive transmission in all six datasets and pick one from each type (hilly and mountainous) to plot the points transmitted and render the reconstructed terrain. Figure. 6.2 and Figure. 6.3 are from data sets hill2 and mtn1.

In Table 6.2, we give the number of points and compressed size in different stages of the progressive transmission. The process begins with 160 initial points selected by TIN. Points are selected afterwards according to the absolute elevation
Table 6.2: Progressive Transmission: Compressed size of points sent are shown, with the corresponding error given as the column title. Algorithm ends when RMS error of the reconstructed surface < 10. Compressed results of our method and Bzip2 are compared, RLE+LP means Runlength encoding for \((x, y)\) and linear prediction for \(z\).
ends when the RMS error is smaller than 10. The sizes after further compression are recorded. The Table 6.2 values are also plotted in Figure 6.4.

We also show in Figure 6.5 the relationship between compressed size and RMS elevation error of the surface reconstructed by ODETALP. The compressed size corresponds to Table 6.2, and the RMS error of the reconstructed surface is shown in $y$ axis. We see that when 1KB of compressed point file are sent, the RMS error would drop by almost 50% and when 2KB more are sent, the RMS error will be smaller than 20.
Figure 6.4: Compressed size at different stages of compression for both our method and bzip2. We list 5 stages: when initial points are selected, when RMS error drops below 75%, 50%, 25% of the initial RMS error for the first time and when RMS error is smaller than 10.

Figure 6.5: Root-Mean-Square error of reconstructed surface is plotted against the size of compressed points. As before, we use run length encoding and linear prediction to do the compression.
7. CONCLUSION

We describe a new terrain compression technique based on an Overdetermined Laplacian equation that achieves good compression ratio and high quality. Using the ODETLAP method, it is possible to achieve a lossy compression with compressed size of 1% to 3% of the original binary file while keeping a reasonable error. This is highly useful when compression ratio is emphasized over accuracy. We use several different points selection methods as to find most important points. In order to maximize compression ratio, we use a forbidden zone in selecting refined points as well as run length encoding and linear prediction in compression of all \((x, y, z)\) points.

In addition to the new representation itself, we also present some application of the new method. A progressive transmission method is given based on the ODETLAP lossy compression technique, which improves the performance of terrain image transmission. Since the data being transmitted is no longer the whole terrain but a small subset of points, it is possible to achieve greater transmission throughput. We also introduce the application of ODETLAP in hydrology. ODETLAP is used to reconstruct the terrain while important features like visibility and hydrology information are preserved.
8. FUTURE WORK

The next research step consists of a few extensions in several directions: We may investigate other PDEs to see if they can reconstruct the terrain more accurately than the Laplacian PDE. Since currently we use lossless compression in the final compression step, we will test the use of lossy schemes as well, which can reach higher compression ratio. Parallel ODETLAP is also a direction of research. Currently, ODETLAP’s processing capability is limited to terrain images of a few hundred by a few hundred. The parallel approach will probably works as follows: First divide the terrain into manageable patches, for each patch create a process/thread to do ODETLAP on that patch. When all processes end, merge all them together into a complete terrain. There are also other ways to do the parallel implementation, such as multi-grid method and use parallel QR factorization to solve ODETLAP. The goal is to improve the scalability and efficiency of ODETLAP.
LITERATURE CITED


PARALLEL TERRAIN COMPRESSION AND
RECONSTRUCTION

By

Jared Stookey

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
Major Subject: COMPUTER AND SYSTEM ENGINEERING

Approved:

W. Randolph Franklin, Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

February 2008
(For Graduation December 2008)
LIST OF TABLES

2.1 Three mountainous terrain data sets are compressed with ODETLAP. The table shows: elevation range and standard deviation, original and compressed size, compression ratio, number of points selected, elevation error, elevation error as percent of elevation range, and slope error. . . 7

6.1 Comparison of different patch sizes: Starting with original terrain of size 2000 \times 2000, input points are sampled every 10 points in the x and y directions, thereby selecting a total of 1% of the total number of points. A patch size of 100 \times 100 gives a good compromise between running time and accuracy. Reported errors in these examples are with respect to the original terrain data set. . . . . . . . . . . . . . . . . . . 25

6.2 This companion data to Table 6.1 reports the errors of Patch ODETLAP with respect to Non-Patch ODETLAP processing of the same test data consisting of 800 \times 800 points. Note that the parallel version introduces only a very small amount of error. . . . . . . . . . . . . . . . . . . 27

6.3 Comparison of three versions of ODETLAP: non-patch, serialized, and parallel on an 800x800 data set. Reported errors are with respect to the original terrain data set. . . . . . . . . . . . . . . . . . . 27
LIST OF FIGURES

1.1 The parallel version of ODETLAP was used to generate a dense 10000 × 10000 raster (B) from raw urban LIDAR data consisting of 13 million points (A) quickly on a cluster of 128 2.6 GHz AMD Opteron processors. 2

1.2 ODETLAP is applied to overlapping layers of patches. In this example, 4 patches would be enough to cover the entire heightmap, but we would see errors at the patch edges. Instead, ODETLAP is run on 9 overlapping patches. 3

1.3 This figure shows the layout of the overlapping patches. This example shows a 400x400 grid with 49 (7x7) 100x100 patches. 3

1.4 Our final step is to merge the overlapping patches (A) into the complete reconstructed elevation map (B). 4

3.1 ODETLAP performance on a single large patch on grid with 1% known (original) points. The “Number of Points” in the x-axis label refers to the number of points in the reconstructed grid. The calculation time grows quadratically with the number of points, and it takes nearly 10 minutes to process an 800 × 800 grid. 9

3.2 Altering a single known point in the terrain has a limited radius of impact during reconstruction. This plot shows the per-pixel difference between the ODETLAP solution (using a 1% subsampling of a 400 × 400 elevation map with elevations ranging from 219 to 1040) and the ODETLAP solution for the same data with the central point edited by setting its elevation to 100,000. The difference in the solutions was restricted to a 62 × 62 area around the altered point, with 88% of the difference concentrated in the center 30 × 30 points. 10

3.3 In the patch ODETLAP method, the compressed terrain (A) is divided into patches (B). Next, ODETLAP is run on each patch individually, which reconstructs a small portion of the entire elevation map (C). Finally, all of the patches are merged into the final approximated solution (D). 11

3.4 Due to incomplete data, ODETLAP will have errors near the borders of a patch when the results from the non-patch ODETLAP method are compared with the results from running ODETLAP on a small patch. The error plot on the right shows correct results in blue, and elevation differences of 5 or more units (meters) in red. 12
3.5 Poor results are seen if the patches are naively merged. A) shows discontinuities in the naively merged elevation map, and B) shows the errors.

3.6 A simple averaging of overlapping patches reduces some of the border error of the reconstructed terrain. A) shows the image that has been merged by averaging, and B) shows the errors. Notice the visible discontinuities where patch edges are averaged.

3.7 A point $p_i$ is shown with respect to its patch. The distance from the point to the edges of the patch in each dimension are used to calculate the weight when doing bilinear interpolation to merge multiple patches together.

3.8 Bilinear interpolation (left) is used instead of a simple averaging (right) to do a weighted averaging of four pixels to merge four patches. Note that the border patches are a special case, where fewer than four pixels are merged.

3.9 Bilinear interpolation is used to do a weighted average such that border values fall off to zero. This results in a visibly continuous reconstructed image (A), and small error values (B), when compared with results from running the non-patch version of ODETLAP on the same terrain.

4.1 A 2000 × 2000 grid is reconstructed from a subset of 1% of the original points by running ODETLAP on a distributed platform consisting of a cluster of AMD Opterons running at 2.6GHz.

5.1 This figure shows the layout of the data blocks. This example shows a 400 × 400 grid with 100 × 400 blocks and 100 × 100 patches.

6.1 This 16000 × 16000 elevation matrix can be processed by parallelized ODETLAP in 28 minutes and 32 seconds.

6.2 The original terrain (A) is compared to terrain that has been compressed by sampling on a regular grid, and then reconstructed using the patch method (B). Notice that some smoothing has occurred.

6.3 A difference map that shows the error between (A) and (B) from Figure 6.2. The error has a range of 0.29.

6.4 A close-up of a particularly challenging 100 × 100 section taken from the grid in Figure 6.1.

6.5 Execution time for the non-patch version of ODETLAP grows quadratically with the number of pixels, while the patch version of ODETLAP has a linear growth. This improvement is gained while running serially.
ACKNOWLEDGMENT

Without the support of the people close to me, I would never have been able to have accomplished so much over the last few years. First and foremost, I would like to thank my wife, Mindy, who has persevered through endless hours of giving me space to get work done, and always being there to encourage me when it was needed.

I would like to extend my deepest gratitude to the members of the Geostar research project for getting me involved in the research community, and always offering valuable advice. I would like to thank my advisor, Professor W. Randolph Franklin, and Professor Barb Cutler for their insights and constant encouragement. I would like to thank Zhongyi Xie, who worked with me on this project and was instrumental in the process of designing, implementing, and documenting the parallel ODETLAP method. Many parts of this thesis would not have been possible without his help, and Zhongyi collected the data in the three tables in this thesis.

I would like to thank Dan Tracy, who is responsible for laying the groundwork for the patch-based ODETLAP idea. Also, my thanks go out to Jon Muckell, Chris Stuetzle, Metin Inanc, and Marcus Andrade for welcoming me into the group and for their willingness to share new ideas.

I cannot begin to thank my co-workers in the ECSE department enough for making it possible to work towards a Masters degree. A.J. Lacomba helped me in countless ways, and always encouraged me to do my very best and for that I am very grateful. My thanks go out to Priscilla Magilligan, who walked me through every step of the masters program. I would also like to thank Mick Neal, George Narode, Ann Bruno, Ken Connor, and Kim Boyer for their continuous support.

Lastly, I would like to thank my grandfather, Stanley Donald Stookey, who has always been there for me, and continues to be my biggest inspiration to work hard, be generous, and enjoy life.
ABSTRACT

We introduce a parallel approximation of a solver for an Over-Determined Laplacean system of linear equations (ODETLAP) which is applied to the compression and restoration of terrain data used for Geographical Information Systems (GIS). ODETLAP can be used to reconstruct a compressed elevation map, or to generate a dense regular grid from airborne Light Detection and Ranging (LIDAR) point cloud data. With previous methods, the time to execute ODETLAP does not scale well with the size of the input elevation map, resulting in running times that are prohibitively long for large data sets. The algorithm presented here divides the data set into patches, runs ODETLAP on each patch, and then merges the patches together. This method gives two distinct speed improvements. First, scalability is provided by reducing the complexity such that the execution time grows almost linearly with the size of the input, even when run on a single processor. Second, ODETLAP can be calculated on the patches concurrently in a parallel or distributed environment. This new patch-based implementation takes 2 seconds to run ODETLAP on an $800 \times 800$ elevation map using 128 processors, while the original version of ODETLAP takes nearly 10 minutes on a single processor (271 times longer). The effectiveness of the new algorithm is demonstrated by running it on data sets as large as $16000 \times 16000$ on a cluster of computers. Results from running on an IBM Blue Gene/L system with 32,768 processors are also discussed.
1. Introduction

Due to recent advances in GIS data acquisition methods such as LIDAR and satellite imagery, vast amounts of GIS data is being collected at such a fast rate that storing and processing all of this data currently poses a problem, and will increasingly pose a problem in the near future. Research can help to reduce these problems in several ways. First, the size of the stored data can be reduced by inventing new compression techniques that are optimized for GIS data. Second, data can be processed more quickly using new approximation techniques that will reduce the time required for operating on larger data sets. Third, by taking advantage of machines such as multi-processor and multi-core machines, as well as computing clusters and supercomputers, it will be possible to transform vast amounts of raw data collected in the field into strategic planning intelligence in a short amount of time. In this thesis, a method to accelerate the process of ODETLAP terrain compression and reconstruction [12] that demonstrates each of these will be introduced.

ODETLAP has proven to be an effective method for generating terrain that closely represents the original data by filling in the unknown points. This feature can be used to generate raster data from sparse point cloud data, or to reconstruct an elevation map from a small subset of points. Using ODETLAP as a black box, we have developed terrain compression algorithms to reduce the storage space for large terrains. However, ODETLAP suffers from a scalability issue. As the dimensions of the elevation map increase, the numerical complexity of ODETLAP causes computation time to increase quadratically with the number of pixels (Figure 3.1). In this thesis, we describe an approximation for ODETLAP in which the terrain is divided into overlapping patches which ODETLAP can process quickly. Figures 1.2 and 1.3 show how the individual patches are positioned, and how they overlap. After ODETLAP is calculated for a patch, the patch is combined (Figure 1.4) with the others in a process called merging. This method, called patch-based ODETLAP, quantizes the numerical complexity of ODETLAP, so that the execution time grows linearly with the size of the input matrix. Also, by dividing the input matrix into
Figure 1.1: The parallel version of ODETLAP was used to generate a dense $10000 \times 10000$ raster (B) from raw urban LIDAR data consisting of 13 million points (A) quickly on a cluster of 128 2.6 GHz AMD Opteron processors.

In order to achieve the best results for different data set sizes and different types of computer systems, two different algorithms are used. The algorithms trade off between the maximum size of data sets that can be processed and the total execution time. The first algorithm assumes that the entire data set fits in the computer’s memory. By making this assumption, no cache is required, and Parallel ODETLAP can operate at its fastest. In order to overcome this limitation, we propose a second method to use the disk as a cache. This method is slower, but the maximum data set size is greatly increased.

Figure 1.1 provides a powerful demonstration of ODETLAP’s ability to fill in unknown points. Given over 13 million sparse point cloud data points generated using airborne LIDAR, ODETLAP generates a very accurate representation of the
Figure 1.2: ODETLAP is applied to overlapping layers of patches. In this example, 4 patches would be enough to cover the entire heightmap, but we would see errors at the patch edges. Instead, ODETLAP is run on 9 overlapping patches.

![Diagram showing overlapping patches]

Figure 1.3: This figure shows the layout of the overlapping patches. This example shows a 400x400 grid with 49 (7x7) 100x100 patches.

original urban elevation map. Using the parallel patch-based ODETLAP, this calculation took 33 minutes and 22 seconds on 128 processors in a cluster of 2.6 GHz AMD Opterons. Extrapolating from the data in Figure 3.1, the same calculation would have taken more than 179 days using the non-patch version of ODETLAP on a single processor.

Parallel ODETLAP greatly improves the efficiency and can still keep up with the non-patch ODETLAP in reconstruction accuracy. Because generous overlap between the patches is provided, and it is ensured that sufficient known data samples are shared between the patches, the result of the patch-based parallel ODETLAP matches the non-patch version of ODETLAP within 0.1%.

The contributions are as follows:
Figure 1.4: Our final step is to merge the overlapping patches (A) into the complete reconstructed elevation map (B).

- A partitioning scheme for calculating ODETLAP that greatly reduces the overall numerical complexity for large elevation maps.
- A method for distributing and merging data partitions in order to calculate ODETLAP in a parallel or distributed environment.
- Demonstration of an algorithm for calculating ODETLAP on much larger terrains than what was previously possible with almost identical results.
- A method for loading and saving very large elevation maps in which the data is streamed between memory and the disk such that a data set which is larger than the available memory can be operated on.
- Multiple methods of running ODETLAP in a distributed environment that optimize for:
  - Multi-processor systems with large memory
  - Cluster of systems with large memory
  - Distributed systems with small memory but many CPU’s:
    1. Very fast execution time on data that can fit in one CPU’s RAM
    2. Slower execution time, but capable of running on much larger data
2. Related Work

Research on parallel computing in GIS began to play an important role in the 1990s [1] as parallel computing technology became more widely available. Research at the Edinburgh Parallel Computing Centre (EPCC) emphasized creating new parallel libraries to support high performance GIS data models [1, 6, 10]. A newer approach to the problem of parallelizing GIS operations was taken in Huang’s paper [7] in which the GRASS GIS application was modified to operate in a clustered environment. Ichikawa et al [8] demonstrate an iterative data partitioning scheme for parallelizing a PDE solver. Griebel et al [5] made excellent progress using parallel multigrid to solve PDE’s. More recently, we have seen research in image processing that involves large linear systems on huge images using a streaming multigrid that can benefit by running on parallel systems [9] in such a way that could be adapted to problems in GIS. The parallel ODETLAP method described in this thesis extends [11] to include algorithms for running on much larger data sets on the Blue Gene/L System. The original ODETLAP method that the parallel method described in this thesis is based on was developed by Gousie and Franklin ([4], [3]).

We extend these important pieces of research to the parallelization of over-determined PDE’s. GIS applications are specifically targeted, and we provide an in-depth analysis of the quality-cost trade-off associated with partitioning very large elevation maps. In this thesis, a parallel terrain compression algorithm is proposed that is capable of processing terrain maps of size $16000 \times 16000$ and larger in a reasonable amount of time.

2.0.1 The ODETLAP Solver

The ODETLAP solver takes a sparse set of known elevations as input, and the output is a complete elevation map. In order to calculate the complete elevation map, ODETLAP forms a set of linear equations with the following constraints. First, for all unknown elevations, the elevation is equal to the average of its four neighbors, as expressed by:

$$\frac{1}{2}$$
Second, for all known elevations, the elevation is equal to the known elevation at that point:

\[ z_{xy} = h_{xy} \] (2.2)

These equations form a large over-determined system of linear equations based on Laplace’s equation. Equation 2.1 enforces smoothness between adjacent points. However, Equation 2.2 can overcome this constraint. For example, adjacent points with known elevations may not be equal to the average of the neighbors. For this reason, there is no exact solution, and an approximation is made. An additional parameter \( R \) is added to trade off between smoothness and accuracy of the known points. Equation 2.1 is weighted by \( R \) relative to Equation 2.2. Using a small \( R \) will approximate a determined solution, and the solution will be more accurate with respect to the known elevations. A large \( R \) gives Equation 2.1 more weight and will result in a smoother solution, but some of the known elevations may change. Refer to Xie’s paper [12], and Gousie and Franklin’s papers ([4], [3]) for a more in-depth discussion of ODETLAP.

### 2.0.2 ODETLAP-based Terrain Compression

Since ODETLAP is capable of reconstructing the whole DEM matrix from a few sparse input points \((x, y, z)\), it can be used as a decompressor in the ODETLAP-based compression algorithm. In order to reduce the amount of space required to store the data, only a limited subset of the elevations from the original elevation data are stored. Later, ODETLAP is used to lossily reconstruct the elevation map by filling in the missing points.

The DEM first undergoes a point selection which picks a subset of points, \( S \), as input to the ODETLAP solver. ODETLAP selects points that are deemed important to the accurate reconstruction of the terrain such as contour lines, border points, or any other available points. The points can consist of a sparse point cloud,
Table 2.1: Three mountainous terrain data sets are compressed with ODETLAP. The table shows: elevation range and standard deviation, original and compressed size, compression ratio, number of points selected, elevation error, elevation error as percent of elevation range, and slope error.

<table>
<thead>
<tr>
<th></th>
<th>Mtn1</th>
<th>Mtn2</th>
<th>Mtn3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elev range</td>
<td>1040m</td>
<td>953m</td>
<td>788m</td>
</tr>
<tr>
<td>St. Dev.</td>
<td>146.0</td>
<td>152.4</td>
<td>160.7</td>
</tr>
<tr>
<td>Orig size</td>
<td>320KB</td>
<td>320KB</td>
<td>320KB</td>
</tr>
<tr>
<td>Compr. size</td>
<td>9744B</td>
<td>9670B</td>
<td>9895B</td>
</tr>
<tr>
<td>Compr. ratio</td>
<td>33:1</td>
<td>33:1</td>
<td>32:1</td>
</tr>
<tr>
<td># pts selected</td>
<td>4160</td>
<td>4160</td>
<td>4160</td>
</tr>
<tr>
<td>Elev. RMS error</td>
<td>9.48</td>
<td>9.55</td>
<td>9.68</td>
</tr>
<tr>
<td>Elev. RMS perc.</td>
<td>0.91%</td>
<td>1%</td>
<td>1.23%</td>
</tr>
<tr>
<td>Slope RMS error</td>
<td>8.34°</td>
<td>8.36°</td>
<td>7.87°</td>
</tr>
</tbody>
</table>

2.0.3 Regular and Irregular Point Data

This thesis demonstrates ODETLAP in two different reconstruction scenarios. In Figure 1.1 a LIDAR point cloud with approximately 13 million points is inflated to a 100 million point dense grid. In the other examples in the thesis the terrain is reconstructed from a compressed subset of points consisting of 1% of the total points, uniformly sampled in a regular grid. With the ODETLAP compression scheme the regular grid can optionally be augmented with additional important points to reduce the error in the reconstructed elevation map, however this is not a concern in the context of this thesis because the focus of this thesis is on the parallelization of the algorithm rather than optimizing the compression method. The primary difficulty for using only irregular data with the patch method is ensuring that every patch contains enough data and sufficiently overlaps the data from neighboring patches to accurately reconstruct the data points. This is left as an area for future work.
3. Approach

The time it takes to calculate ODETLAP on an elevation map does not scale as the size of the data set increases, and the execution time quickly becomes prohibitively long. Figure 3.1 illustrates the ODETLAP performance issue. When the terrain is of size $50 \times 50$ and $100 \times 100$, it takes less than one second to run ODETLAP. However, when ODETLAP is run on an $800 \times 800$ terrain, it takes nearly 10 minutes to finish, and we would like to run ODETLAP on terrains of much larger sizes. Thus, the goal is to develop a scalable implementation of ODETLAP to allow manipulation of large terrains, such as those collected by LIDAR scanners.

As the data set size grows, more and more points are involved in the calculation, some of which are quite distant and thus have little influence on each other. Figure 3.2 shows that when a single known point is edited in the input to ODETLAP, only points within a small neighborhood of the point are affected. Beyond that small region, the effect becomes negligible. This supports the hypothesis that it should be possible to divide large data sets into separate patches, run ODETLAP on them individually, and achieve similar results to the non-patch ODETLAP solution. Specifically, a patch size of $100 \times 100$ should be sufficient to capture the detail for a terrain grid with a subsampling percentage of 1%. When the reconstructed patches are merged back together into the reconstructed elevation map as shown in 3.3, the difference between the reconstructed and original should be minimal.

3.0.4 Dividing into Patches

The ODETLAP calculation depends on elevation data from neighboring pixels, which causes a problem when running it on individual patches. At the edges of a patch, there is less information to work with, and therefore the calculations tend to have errors when compared to the non-patched-based ODETLAP solution. Figure 3.4 highlights the problem by showing that when the image is reconstructed, the values are generally correct near the center of the patch, but near the edges of the patch, it is common to encounter errors of 5 or more.
Figure 3.1: ODETLAP performance on a single large patch on grid with 1% known (original) points. The “Number of Points” in the x-axis label refers to the number of points in the reconstructed grid. The calculation time grows quadratically with the number of points, and it takes nearly 10 minutes to process an $800 \times 800$ grid.

To determine the value for the error, ODETLAP is calculated on the heightmap using a single large patch that covers the entire elevation map, and then ODETLAP is calculated on each individual patch. The error is defined as the absolute value of the difference between the corresponding elevation values for each method. If the terrain map were simply divided into a single layer of non-overlapping patches, then reconstructed with ODETLAP, and joined at their edges to form the reconstructed image, then we would see results like the ones in Figure 3.5. At the patch borders, there are areas of large error and drastic discontinuities.

To avoid errors at the borders of the patches, ODETLAP is run on overlapping layers of patches. Figure 1.2 illustrates an example where instead of dividing an elevation map into 4 non-overlapping patches, the elevation map is divided into 9 overlapping patches. Figure 1.3 shows a more realistic case where a $400 \times 400$ terrain is divided into 49 $100 \times 100$ patches.
Figure 3.2: Altering a single known point in the terrain has a limited radius of impact during reconstruction. This plot shows the per-pixel difference between the ODETLAP solution (using a 1% subsampling of a 400 × 400 elevation map with elevations ranging from 219 to 1040) and the ODETLAP solution for the same data with the central point edited by setting its elevation to 100,000. The difference in the solutions was restricted to a 62 × 62 area around the altered point, with 88% of the difference concentrated in the center 30 × 30 points.

3.0.5 Merging the Patches

After ODETLAP is calculated on multiple patches, the patches are merged into the final result. The basic idea is shown in Figure 1.4. There are many possible ways to go about merging the patches. The overly simple method of uniformly averaging the values of all contributing patches will still contain discontinuities and large errors at the patch edges as seen in Figure 3.6. However, because ODETLAP is calculated redundantly for each pixel, the erroneous edge pixels can be ignored by selecting those values from the center of a different patch.

Instead of a simple averaging method, bilinear interpolation is used, which gives a greater weight to pixels near the center, and the weight falls off to zero for
Figure 3.3: In the patch ODETLAP method, the compressed terrain (A) is divided into patches (B). Next, ODETLAP is run on each patch individually, which reconstructs a small portion of the entire elevation map (C). Finally, all of the patches are merged into the final approximated solution (D).

Pixels near the edges. Using the following definitions as shown in Figure 3.7:

- $p$: elevation of a point that we wish to interpolate from the elevations of 4 contributing, overlapping patches
- $i$: contributing patch number
- $P_w$: patch width
- $P_h$: patch height
- $p_i$: elevation of the point of interest in patch $i$
- $d_{xi}$: distance from $p_i$ to the nearest patch edge along $x$
- $d_{yi}$: distance from $p_i$ to the nearest patch edge along $y$
- $w_i$: weight given to $p_i$

We can use equations 3.1 and 3.2 to find the elevation for a point $p$. First, we calculate the weight to apply to the elevation of a point in a patch:
Figure 3.4: Due to incomplete data, ODETLAP will have errors near the borders of a patch when the results from the non-patch ODETLAP method are compared with the results from running ODETLAP on a small patch. The error plot on the right shows correct results in blue, and elevation differences of 5 or more units (meters) in red.

\[ w_i = \frac{p_d x_i d_y_i}{P_x^2}, \quad (3.1) \]

Next, we apply the weights to each point, and add the contribution to the point \( p \) in the full elevation map:

\[ p = \sum_{i=1}^{4} w_i p_i, \quad (3.2) \]

Note that this formula applies to all of the points except points on the corners and edges of the full elevation map. For points in the corner of the elevation map in a section of size \( \frac{P}{2} \times \frac{P}{2} \), only a single patch contributes, so there is no interpolation. For edge pixels between the corner patches, we interpolate between only 2 contributing patches, in which case we interpolate only in one dimension. Equation 3.1 is modified by removing the entries associated with the dimension that can not be interpolated (either \( d_x \) and \( \frac{P}{2} \) or \( d_y \) and \( \frac{P}{2} \)), then changing the 4 to a 2 in Equation 3.2.

A visualization of the weights can be seen in Figure 3.8. The image on the left shows the bilinear interpolation weighting pattern for a single patch. The image on the right shows the weighting pattern using simple averaging. Red pixels represent a weight of 1.0, and blue pixels represent a weight of 0.0. In both weighting methods,
Figure 3.5: Poor results are seen if the patches are naively merged. A) shows discontinuities in the naively merged elevation map, and B) shows the errors.

when four patches are merged, the sum of all of the weights for a given pixel is one. Using bilinear interpolation, the patches are merged such that the pixels at the patch’s borders are ignored, but the correct pixels near the center contribute most of the value. This results in a reconstructed elevation map that has very small errors, and no visible discontinuities, as seen in Figure 3.9.

Because non-patch ODETLAP is prohibitively slow on large data sets, the error analysis and plots shown in Figures 3.4, 3.5, 3.6, and 3.9 were performed on 400×400 terrain data. The elevations for this DEM range from 1105 to 1610 meters, and is represented as integer values with units of one meter.
Figure 3.6: A simple averaging of overlapping patches reduces some of the border error of the reconstructed terrain. A) shows the image that has been merged by averaging, and B) shows the errors. Notice the visible discontinuities where patch edges are averaged.

Figure 3.7: A point $p_i$ is shown with respect to its patch. The distance from the point to the edges of the patch in each dimension are used to calculate the weight when doing bilinear interpolation to merge multiple patches together.
Figure 3.8: Bilinear interpolation (left) is used instead of a simple averaging (right) to do a weighted averaging of four pixels to merge four patches. Note that the border patches are a special case, where fewer than four pixels are merged.

Figure 3.9: Bilinear interpolation is used to do a weighted average such that border values fall off to zero. This results in a visibly continuous reconstructed image (A), and small error values (B), when compared with results from running the non-patch version of ODETLAP on the same terrain.
4. Implementation for Large Memory Clustered Systems

Our first efforts were to parallelize ODETLAP on clustered systems with enough memory that a single process can hold the entire source or result in memory at one time. In this version, the merging process requires that the merging process store the entire elevation map in memory. The patch version of ODETLAP was implemented using MPI, which is capable of running the software on parallel and distributed platforms. When MPI starts, the first process is assigned the task of waiting for reconstructed patch data from the rest of the processes, which are designated as worker processes. All of the workers are pre-assigned a set of patches to process. Each patch overlaps in both the x and y directions, as seen in Figure 1.3. For each assigned patch, the worker process does the following:

- Load the patch
- Run ODETLAP on the patch
- Send the reconstructed patch to the central process

As the patches are collected by the first process, the values are weighted and merged into the full reconstructed elevation map, which is then saved to the hard disk.

A simple direct linear system solver would have required cubic time to execute. However, the implementation for this thesis was built on the QR solver from the CSparse library [2]. The execution time is nearly quadratic with respect to the number of points in a single patch, and in the case of the non-patch ODETLAP version, with respect to the number of points in the entire input elevation map.

Parallelized patch ODETLAP was implemented on a cluster of 2.6 GHz AMD Opteron machines running Red Hat Enterprise Linux 4.5. The results can be viewed in Figure 4.1. This method had some limitations that needed to be overcome in order to optimize for the Blue Gene/L System (see Chapter 5). In this implementation, the size of the input elevation map is limited by the amount of memory required to store the entire grid while merging it. Also, there is a bottleneck when reading the
Figure 4.1: A 2000 × 2000 grid is reconstructed from a subset of 1% of the original points by running ODETLAP on a distributed platform consisting of a cluster of AMD Opterons running at 2.6GHz. For this implementation, a linear decrease in running time was achieved for up to 128 processors, but after that, the overhead of file I/O prevents any additional speedup without further optimizations. The algorithm includes a central process to merge results, but this process is not included in the x axis.

The input data set is loaded once for every patch that needs to be calculated. For example, when calculating ODETLAP on a 16000 × 16000 grid, there are 101761 patches, and the file is loaded once for each of those patches. When more processors are used, it means that more processors are all trying to read the file simultaneously, limiting the method’s scalability.
5. Blue Gene/L System Implementation

The Blue Gene/L System presented additional challenges beyond those imposed by a cluster of computers with large memory. The Blue Gene/L supercomputer at Rensselaer Polytechnic Institute has access to 32,768 processors, each with two cores. Each processor has access to 1 Gigabyte (G) of memory. While it is possible to share 1G of memory between 2 cores by running in coprocessor mode, this is not the most efficient way to use the Blue Gene, because only one of the cores can then run as a worker process. It is more efficient to use both cores as independent virtual nodes, each with 512MB of memory. This means that in order to take advantage of all the processors to handle terrain data larger than approximately $15000 \times 15000$ grids of float values, the data needs to be streamed to and from the hard disk as it is operated upon. To handle this, tasks are assigned to each process in the following manner:

- 1 source (reads and distributes the input file).
- 1 sink (receives and merges the completed data blocks).
- 1 coordinator (Co-ordinates communication between sink/source and workers, maintains state for each worker).
- (NProcessors - 3) workers.

The input to this version of ODETLAP is the input data file, the output file dimensions, the patch size, and the block size. Data is loaded and communicated in blocks, and ODETLAP is calculated on patches within the data blocks. It was found that a relatively small patch size works well, but for loading and storing the data, it is much more efficient to use the largest data chunks that will fit into memory. For that reason, we use data blocks for transferring data, and we break the blocks up into patches for running ODETLAP on. The algorithms for each process are as follows:
Figure 5.1: This figure shows the layout of the data blocks. This example shows a $400 \times 400$ grid with $100 \times 400$ blocks and $100 \times 100$ patches.

5.0.6 Source Process Algorithm

1. Generate an index of the input file, validate the input data file.

2. For each data block.
   - Allocate memory for the block.
   - For each row of pixels in the block:
     - Seek the 1st entry for current block’s row in the diskfile.
     - Fetch the row of pixels associated with the block.
   - Request the next worker from the coordinator process (else sleep).
   - Send the data block to the worker.
   - Free the memory for the loaded block.

5.0.7 Coordinator Process Algorithm

The coordinator maintains 2 queues of available worker process id numbers:

1. QRecv - A queue of workers waiting to receive the next block from the source.

2. QSend - A queue of workers waiting to send a block to the sink process.

The coordinator process waits for messages from any of the processes, and responds in the following manner:
1. SourceGetNextAvlWorker - if QRecv is empty, then send a message to the requester that no workers were available. Otherwise, pop the next worker off QRecv and send the id to the requesting source process.

2. WorkerWaitingToRecv - Push the worker’s id onto QRecv.

3. WorkerWaitingToSend - Push the worker’s id onto QSend.

4. SinkGetNextSendingWorker- If QSend is empty, then send a message to the requester that no workers were available. Otherwise, pop the next worker off QSend and send the id to the requesting sink process.

5.0.8 Sink Process Algorithm

1. Generate a binary file of size NxN on disk with all zeros.

2. While there are still more blocks to collect:
   
   • Fetch the next completed block number from a worker (else sleep).
   • Request the block from the worker.
   • Receive the block from the worker.
   • Read the entire block from the output file.
   • Merge the worker data with the file data.
   • Save the entire block back to the disk.

5.0.9 Worker Process Algorithm

Loop until there are no more source blocks:

1. Send message to coordinator WorkerWaitingToRecv.

2. Receive a block.

3. Run ODETLAP on each patch in the block and merge the result into a block.

4. Send message to coordinator WorkerWaitingToSend.

5. Receive message that it is ok to send.
6. Send the block to the sink.

There were areas where this method was successful, and other areas where the method was not successful. The algorithm does an excellent job of staying coordinated and communicating status efficiently. The primary bottleneck is the time it takes to access the disk. The worker processes idle while waiting for the sink process to finish writing to the disk. This method distinguishes between patch size and block size, where the patch size represents the size of data that ODETLAP is calculated on, and the block size is the size of the data to be read from and written to the disk. Ideally, the block size should be the maximum amount of data that can be loaded into a single processor’s memory at one time. One problem that we ran into while implementing this method is that the extra abstraction of blocks adds to the complexity of the implementation. The implementation for the original patch method is very straightforward. Adding the extra communication between a sink, source, workers, and the communicator process also leads to a very clean design. However, by adding the abstraction of data blocks, the complexity started to get to be too much to handle, while still remaining understandable. The block method increases the speed of the execution, but does not remove the bottleneck caused by the slow speed of accessing the disk. In order to write the data to the disk, the algorithm requires many random accesses to the output file. The entire output file is written to and read from the disk multiple times. The output file should be written to only one time, and it should be streamed directly to disk without random accesses. In order to accomplish this, the disk should not be used as a cache. Instead, processes need to be allocated to act as data storage. In order for the patch/block system to be fast enough to keep up with the worker processes, it will be necessary to create a robust and reliable library for the different data abstractions. First, spanning the data across multiple processes needs to be implemented. Also, the block system must be implemented very cleanly. The goal should be to allow the main program to depend on reliable block and cache libraries so that the details of these processes do not complicate the main task, which is to divide, calculate, and merge the patches.
6. Results

The parallelized ODETLAP was tested on a 16000 × 16000 DTED Level 2 data set covering roughly 400,000 square miles of the Central USA, primarily in Kansas and Nebraska. The terrain is divided into 101761 patches, each of 100 × 100 size. 128 processors on a cluster of 2.6 GHz AMD Opterons were used and the entire computation took 28 minutes and 32 seconds. The full terrain is shown in Figure 6.1. The range (highest minus lowest) of the test data is 1013 meters and the standard deviation is 217 meters. Every 10 sample points are selected in both x and y dimensions to generate a regular grid input points consisting of 1% of the total points.

Compared to the original terrain, the reconstructed terrain has mean absolute error of 1.96, max absolute error of 50, and root mean square error of 2.76. Note that it is impossible to run the non-patch version of ODETLAP on this large data set, so these results are compared to the original terrain map. A comparison of the original and reconstructed terrain is given in Figures 6.2, 6.3, and 6.4. The two images in Figure 6.2 correspond to the 1000 × 1000 patch in the top left corner of the terrain in Figure 6.1. We can see in the figure that the reconstructed terrain is only losing some high frequency details. A difference map between the two elevation maps (Figure 6.3) shows which areas perform well, and where smoothing occurs. Figure 6.4 presents a close-up view of the area with the highest error. The largest errors occur due to the presence of many extreme variations in elevation within a small (100 × 100) area. These details can be captured more accurately by iteratively adding additional points as described in the previous work for compression using ODETLAP [12].

It is important to note that for the results in this thesis, the focus has been on reconstructing elevation maps from a regular grid of points. The only exception is the LIDAR example in Figure 1.1. While it is possible to run the parallel version of ODETLAP on subsampled points that are irregular, there is no guarantee that the patch method will be able to provide correct results when reconstructing highly
This 16000 × 16000 elevation matrix can be processed by parallelized ODETLAP in 28 minutes and 32 seconds.

sparse patches.

The impact of using different patch sizes on both running time and reconstruction accuracy was examined. In Table 6.1, patch sizes from 20 × 20 to 400 × 400 were used, and the running time, mean absolute error, maximum error, and RMS error of the reconstruction were recorded. We can see that using a patch size of 100 × 100 gives a good balance between accuracy and speed. Table 6.2 shows the small amount of error when comparing the patch version of ODETLAP to the non-patch version. By examining the amount of error introduced versus the overall speedup that is gained, using patches that are 100 × 100 in size produces very good results.

The new patch-based ODETLAP algorithm provides two performance improvements. The first is from decomposing large-scale terrain data into small patches and running ODETLAP sequentially on each of them, which is called serialized ODETLAP. Figure 6.5 shows the running time comparison of the non-patch version
Figure 6.2: The original terrain (A) is compared to terrain that has been compressed by sampling on a regular grid, and then reconstructed using the patch method (B). Notice that some smoothing has occurred.

The second speedup occurs as a result of running ODETLAP on multiple patches concurrently in a parallel environment. Figure 4.1 shows the total running time for a test case with 1521 patches when run on various numbers of processors.

We see that parallelism provides an excellent speedup using up to 127 worker processes. Beyond that, the overhead of parallelism becomes significant. There are opportunities to improve the performance even further. We will need to look closely at ways to optimize disk I/O, which is currently the primary bottleneck. These improvements will be discussed in more detail in Section 7.

Table 6.3 presents the running time and accuracy information for all three ODETLAP versions. The size of the input terrain data is 800 × 800, and the mean elevation is 107. We can see that the patch method only increases errors by approximately 0.1% when compared with the non-patch method, and the running time is reduced to about 0.2% of the original.
Figure 6.3: A difference map that shows the error between (A) and (B) from Figure 6.2. The error has a range of 0..29.

<table>
<thead>
<tr>
<th>Patch Size</th>
<th>Time</th>
<th>Mean Error</th>
<th>Max Error</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 × 20</td>
<td>0m14s</td>
<td>0.6570</td>
<td>13</td>
<td>0.9798</td>
</tr>
<tr>
<td>40 × 40</td>
<td>0m7s</td>
<td>0.6619</td>
<td>13</td>
<td>0.9594</td>
</tr>
<tr>
<td>50 × 50</td>
<td>0m8s</td>
<td>0.6640</td>
<td>13</td>
<td>0.9617</td>
</tr>
<tr>
<td>100 × 100</td>
<td>0m9s</td>
<td>0.6598</td>
<td>13</td>
<td>0.9530</td>
</tr>
<tr>
<td>200 × 200</td>
<td>0m25s</td>
<td>0.6598</td>
<td>13</td>
<td>0.9527</td>
</tr>
<tr>
<td>400 × 400</td>
<td>1m28s</td>
<td>0.6598</td>
<td>13</td>
<td>0.9527</td>
</tr>
</tbody>
</table>

Table 6.1: Comparison of different patch sizes: Starting with original terrain of size 2000 × 2000, input points are sampled every 10 points in the x and y directions, thereby selecting a total of 1% of the total number of points. A patch size of 100 × 100 gives a good compromise between running time and accuracy. Reported errors in these examples are with respect to the original terrain data set.
Figure 6.4: (A) and (B) show a close-up of a particularly challenging $100 \times 100$ section taken from the grid in Figure 6.2. (C) shows a difference map with a range of 0 (blue) to 29 (red). (D) shows the areas with an error larger than 10 in black. Some high-frequency details are lost. Many of these details would have been captured using the more advanced point selection schemes described in [12].
<table>
<thead>
<tr>
<th>Patch Size</th>
<th>Mean Error</th>
<th>Max Error</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 × 20</td>
<td>0.1431</td>
<td>2</td>
<td>0.3801</td>
</tr>
<tr>
<td>40 × 40</td>
<td>0.0532</td>
<td>1</td>
<td>0.2307</td>
</tr>
<tr>
<td>50 × 50</td>
<td>0.0634</td>
<td>2</td>
<td>0.2519</td>
</tr>
<tr>
<td>100 × 100</td>
<td>0.0149</td>
<td>1</td>
<td>0.1223</td>
</tr>
<tr>
<td>200 × 200</td>
<td>0.0039</td>
<td>1</td>
<td>0.0628</td>
</tr>
<tr>
<td>400 × 400</td>
<td>0.0008</td>
<td>1</td>
<td>0.0291</td>
</tr>
</tbody>
</table>

Table 6.2: This companion data to Table 6.1 reports the errors of Patch ODETLAP with respect to Non-Patch ODETLAP processing of the same test data consisting of 800 × 800 points. Note that the parallel version introduces only a very small amount of error.

![Non-Patch ODETLAP vs. Serialized Patch ODETLAP](image)

Figure 6.5: Execution time for the non-patch version of ODETLAP grows quadratically with the number of pixels, while the patch version of ODETLAP has a linear growth. This improvement is gained while running serially.

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Mean Err.</th>
<th>Max Err.</th>
<th>RMS Err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-P ODETLAP</td>
<td>9m8s</td>
<td>0.6150</td>
<td>7</td>
<td>0.8835</td>
</tr>
<tr>
<td>Seri-ODETLAP</td>
<td>0m34s</td>
<td>0.6156</td>
<td>7</td>
<td>0.8846</td>
</tr>
<tr>
<td>Patch ODETLAP</td>
<td>0m2s</td>
<td>0.6156</td>
<td>7</td>
<td>0.8846</td>
</tr>
</tbody>
</table>

Table 6.3: Comparison of three versions of ODETLAP: non-patch, serialized, and parallel on an 800x800 data set. Reported errors are with respect to the original terrain data set.
7. Future Work

The goal of this application is to be able to run ODETLAP on data sets of any size, while making the best use of all of the CPU’s that are available. Currently, the primary bottleneck occurs with the source and sink processes. Only a single source process is used to load the input data from disk, and for that reason the speedup is limited by the amount of time it takes for a single process to read the entire file. For very large data sets, this is a substantial amount of time. There are ways to speed this up. For example, multiple sources could be implemented, and the data could be spanned across multiple files, thereby distributing the process of reading the input. For very large data sets, writing the data out to disk presents more challenges. Data is not streamed directly to the disk as it arrives at the sink. Instead, the existing data block is read from the disk, merged with the incoming data, and written back to the disk again for every data block. This results in reading and writing the file multiple times. This causes a large and unnecessary bottleneck. This was overcome for data sets that fit into memory, but for anything larger than that, another solution is needed. One way to overcome this would be to allocate a group of processes to act as data storage. In practice, inter-process communication on the Blue Gene is extremely fast, so instead of reading and writing data from and to the disk, the data could instead be sent to a process to make use of that process’s memory as a cache. Another possibility would be to change the program so that the blocks are guaranteed to arrive at the sink in order. The sink could then free up the memory for data blocks that have been fully merged. This way, the sink could merge data in memory as it arrives, and then stream the data directly to the disk without ever reading the data back again.

As shown in Figure 3.9, there are still some errors compared with results from running the non-patch version of ODETLAP, which means that improvements may still be possible for the patch merging process to get higher accuracy. This can be combined with strategies to optimize for elevation maps with irregular sampling, such as point cloud data or Triangulated Irregular Network (TIN) data. For exam-
ple, patches with more points should be weighted more heavily than undersampled patches when doing interpolation in the overlapped regions.

Initial test runs have been performed on the 32,768 processor Blue Gene/L system that is part of Rensselaer’s Computational Center for Nanotechnology Innovations (CCNI). With 32,768 processors, the Blue Gene/L system enables us to get much faster performance than what could otherwise be possible. The Blue Gene presents challenges because each processor is limited to between 512 and 1024 MB of memory, and also the individual processor speed is slower than that of a typical processor. The aforementioned improvements to I/O and merging will help to overcome these limitations and enable the parallel ODETLAP implementation to run on data sets with a nearly limitless size in a short period of time.
8. Conclusions

In this thesis we have presented recent progress in parallel terrain compression and reconstruction that can process digital elevation maps of sizes as large as $16000 \times 16000$. ODETLAP is used to reconstruct a terrain map from sparse, isolated samples. Motivated by the fact that the running time of non-parallel ODETLAP increases quadratically with the size of the input data set, the proposed method takes advantage of the fact that local changes in elevation only affect a limited neighborhood. This means that it is not necessary to include the entire large data set into the calculation for every point, so instead the terrain is divided into individual patches such that the size of the patch is large enough to include the entire neighborhood for the pixels near to the center of the patch. The data near the edges of a patch does not include the entire neighborhood, so a system of overlapping patches is implemented and bilinear interpolation is used to select only the accurate data from the center of each patch. This greatly reduces the numerical complexity of the problem, thus increasing the efficiency of ODETLAP. There is also the added benefit that the individual patches can be calculated independently of each other and on multiple processors in parallel. A patch size of 100x100 provided a very good performance gain, while minimally impacting the results of output when compared to the non-patch version of ODETLAP. The results from experiments show that this method greatly reduces the running time and ensures a high quality in the reconstructed image. The best performance gain was achieved by limiting the data set size to that which can fit into memory. By using the hard disk as a cache, this limitation was overcome at the cost of an increased execution time.

When dealing with LIDAR data, the fastest possible turnaround time is desired for collecting the sparse data, converting it to a non-sparse format, processing the elevation data, and acting on the results. The method described in this thesis achieves a tremendous speedup and the ability to process much larger data sets, which brings us much faster overall response time. This new technique changes the way that we will approach large terrains in the future.
REFERENCES


