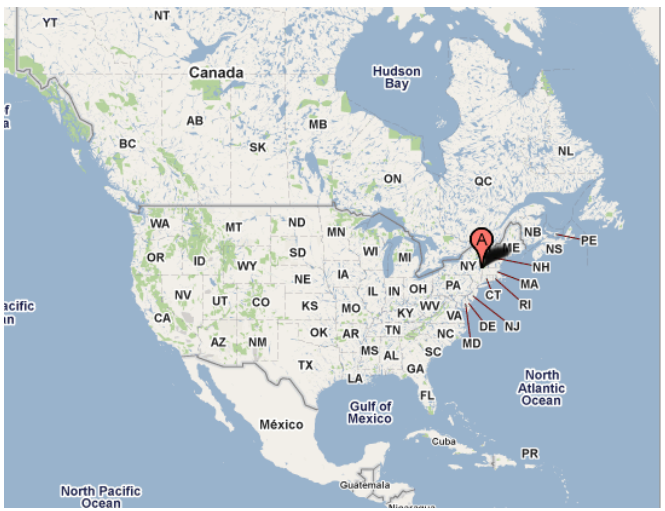
Changing problems, databases, and tools in spatial search

W. Randolph Franklin*, Marcus Andrade*+, Wenli Li*, Salles Magalhães*+

*Rensselaer Polytechnic Institute
Troy, NY, USA

+Federal University of Vi cosa, MG,
Brasil



Dept of Geography, U Zürich, 24 Aug 2015

Outline

My background

- Philosophically a Computer Scientist.
- PhD officially in Applied Math.
- Working in Electrical, Computer, and Systems Engineering.
- Teaching Engineering Parallel Computing.
- Collaborating with Geographers for 45 years.
- Working for Peucker and Douglas, implemented the first Triangulated Irregular Network (TIN) in geography in 1973.
- Enjoy applying computer science and engineering to GIS.

Important concerns in Spatial Search

New:

- problems.
- hardware.
- SW tools.
- algorithms and data structures.

No ArcGIS in this talk

- ESRI is a technological follower.
- I like to do new things.
- They do have some useful tools and make beautiful maps.

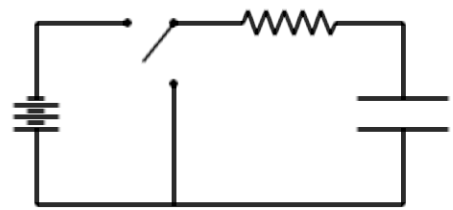
GIS has similarities to parts of Computer Aided Design (CAD).

New search problems

- Coincidences in temporal track logs:
  - past near collisions between aircraft — but note the distance metric.
  - ≥ 5 people from a group of 1000 flocked together.
- Point inclusion against 10^6 polygons — if this is a function in a larger system, then unlikely errors cannot be overlooked.
- Find good observers, or good sites, in $50\,000 \times 50\,000$ map.

Why parallel HW?

- More processing $\rightarrow$ faster clock speed.
- Faster $\rightarrow$ more electrical power. Each bit flip (dis)charges a capacitor through a resistance.
- Faster $\rightarrow$ requires smaller features on chip
- Smaller $\rightarrow$ **greater** electrical resistance !
- $\Longrightarrow\Longleftarrow$.
- Serial processors have hit a wall.



Parallel HW features

- IBM Blue Gene / Intel / NVidia GPU / other
- Most laptops have NVidia GPUs.
- Thousands of cores / CPUs / GPUs
- Lower clock speed 750MHz vs 3.4GHz
- Hierarchy of memory: small/fast $\rightarrow$ big/slow
- Communication cost $\gg$ computation cost
- Efficient for blocks of threads to execute SIMD.
- OS:
  - 187th fastest machine in 6/2013 `top500.org` runs  .
  - 1st through 186th fastest run variants of  .

Massive Shared Memory

- Massive shared memory is an underappreciated resource.
- External memory algorithms are not needed for most problems.
- Virtual memory is obsolete.
- $40K buys a workstation with 80 cores and 1TB of memory.

Code

```
const long long int n(5*000*000*000);
static long long int a[n];
int main() {
double s(0);
for (long long int i=0; i<n; i++)
   a[i] = i;
for (long long int i=0; i<n; i++)
   s += a[i];
std::cout << "n=" << n << ", s="
          << s << std::endl; }
```

Runtime: 60 secs w/o opt to loop and r/w 40GB. (6 nsec / iteration)

Spatial geometric Databases

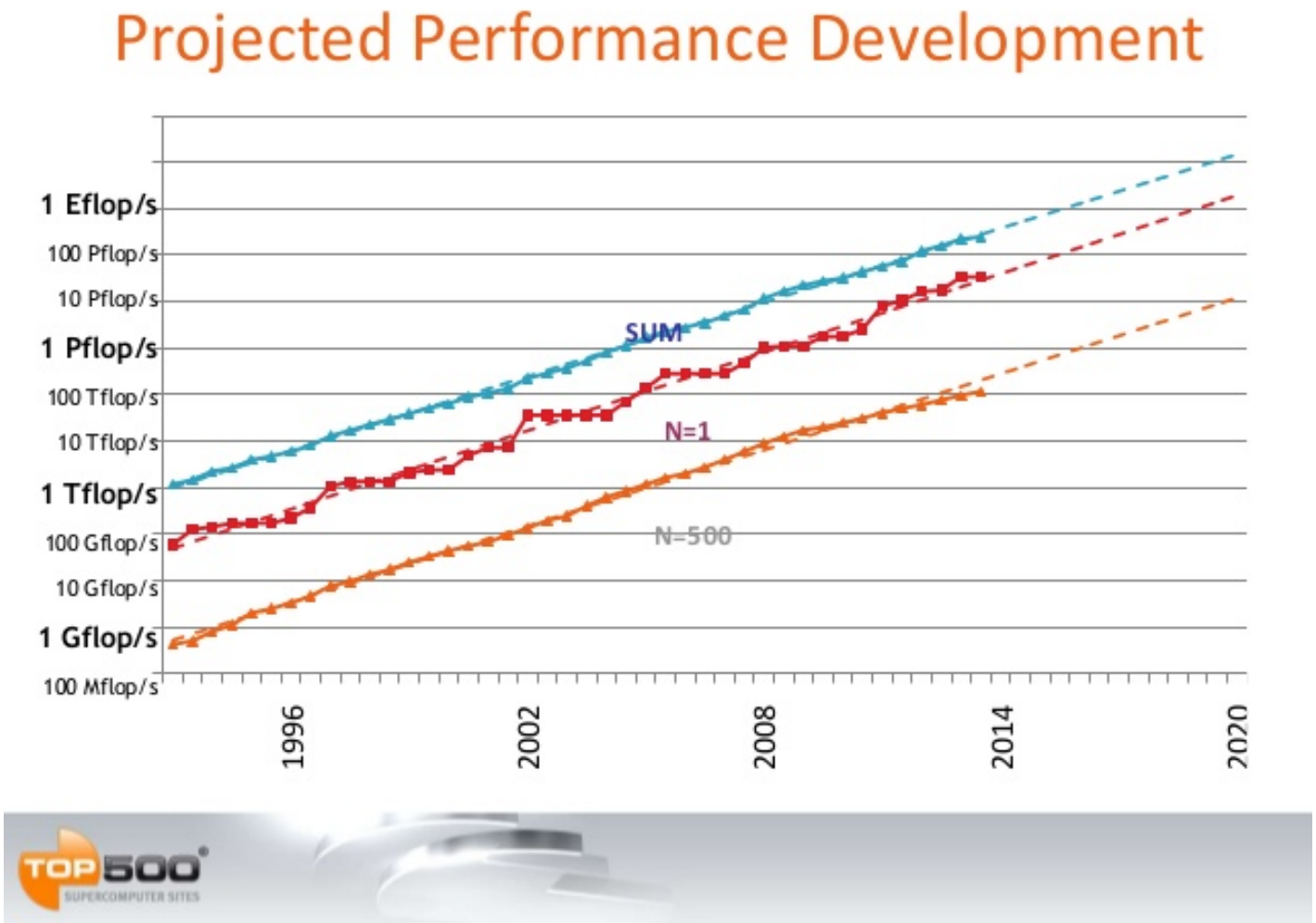
- LIDAR — $10^5 \times 10^5$ terrains.
- New York City taxi logs — 14 million trips in 2013.
- Streaming sensors — process it in real time or lose it.

Outline

Tools

- OpenMP
- CUDA
- Thrust
- gmp++ – big rationals.
- Computational Geometry Algorithms Library (CGAL).
- Matlab
- Mathematica

Commercial tools are expensive.



(Highlights of the 42nd Top500 List, SC13)

OpenMP

- Shared memory, multiple CPU core model.
- Good for moderate, not massive, parallelism.
- Easy to get started.
- Options for protecting parallel writes:
  - Sum reduction: no overhead.
  - Atomic add and capture: small overhead.
  - Critical block: perhaps 100K instruction overhead.
- Cost metric: real elapsed clock time.

OpenMP Example

```
const int n(500000000);
int a[n], b[n];
int k(0);
int main () {
  #pragma omp parallel for
  for(int i = 0; i < n; i++) a[i]=i;
  #pragma omp parallel for
  for(int i = 0; i < n; i++) {
    #pragma omp atomic capture (or critical)
    j = k++;
    b[j] = j; }
  double s(0.);
  #pragma omp parallel for reduction(+:s)
  for (int i=0;i<n;i++) s+=a[i];
  cout << "sum: " << s << endl; }
```

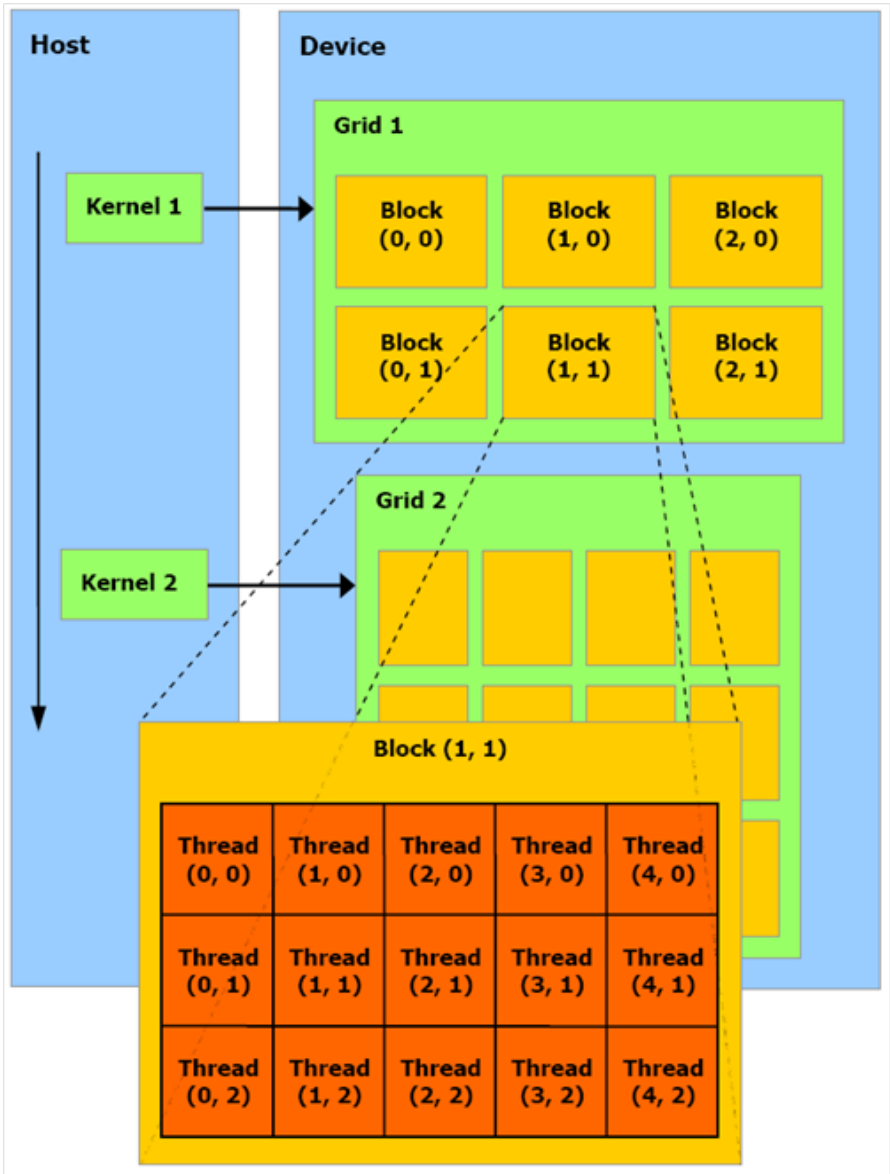
CUDA

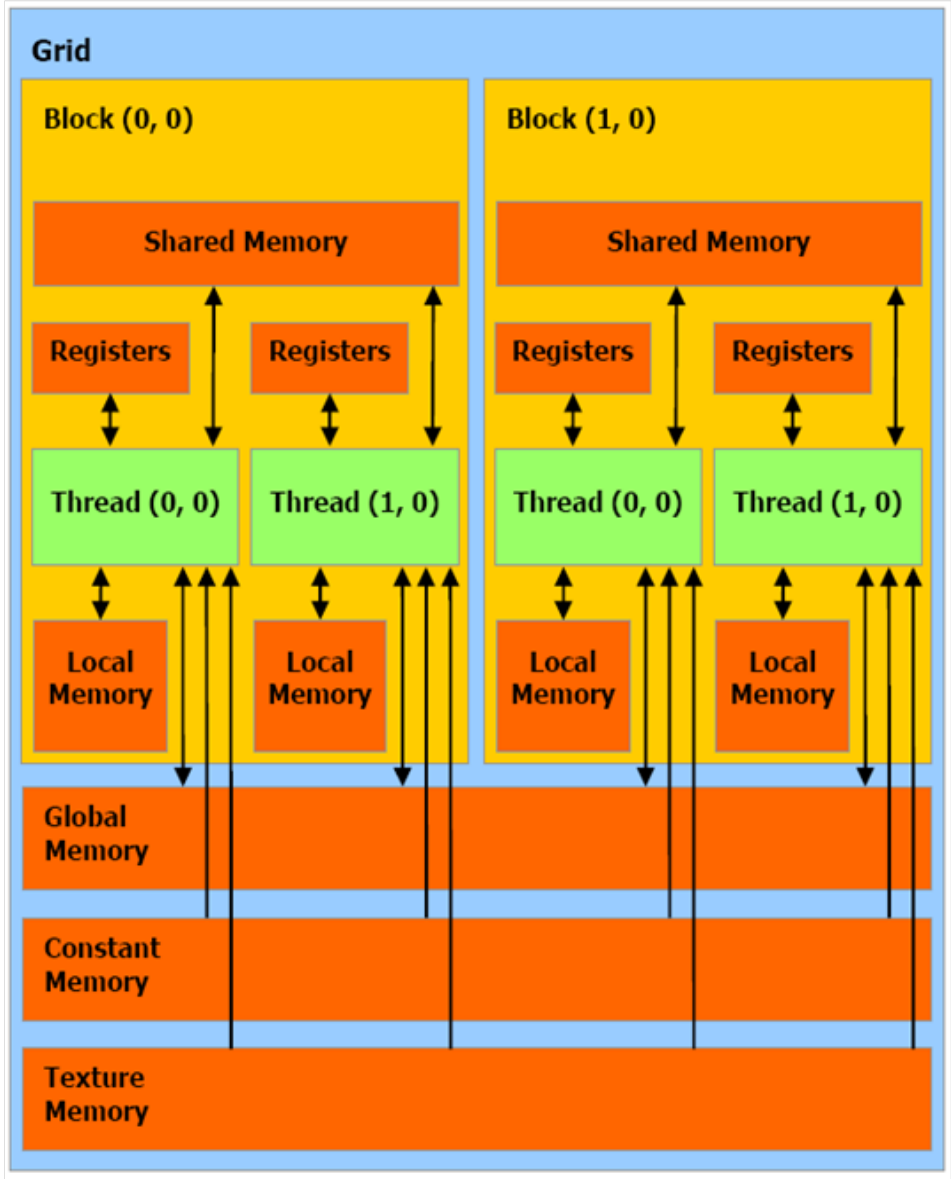
- NVIDIA's parallel computing platform and programming model.
- Direct access to complicated GPU architecture.
- Learning curve: Efficient programming is an art.
- One CUDA core is only 5% as powerful as one Intel Xeon core.
- Following slide from http://www2.engr.arizona.edu/~yangsong/gpu2.png.

Code Fragment

```
__global__ void device_greetings(void)
{
cuPrintf("Hello, from the device!\n");
}
cudaMalloc((void**)&device_array,
  num_bytes)
cudaMemcpy(host_array, device_array,
  num_bytes, cudaMemcpyDeviceToHost);
device_greetings<<<2,3>>>();
```

GPU Architecture





Multiprecision big rationals

- Solves problem of roundoff error when intersecting lines.
- Example uses gmp++ via boost.

Code

```
#include <boost/multiprecision/gmp.hpp>
using namespace boost::multiprecision;
int main() {
  mpq_rational v;
  for(mpq_rational i = 1; i <= 8; ++i) {
    v += (2*i)/(2*i+1);
    std::cout << i << ":  " << v
    << std::endl;
}}
```

Output

```
1:  2/3
2:  22/15
3:  244/105
4:  1012/315
5:  14282/3465
6:  227246/45045
7:  269288/45045
8:  5298616/765765
```

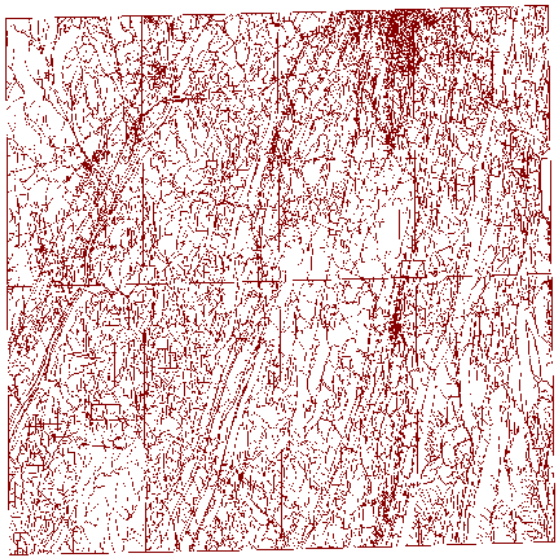
Outline

Algorithm Tactics

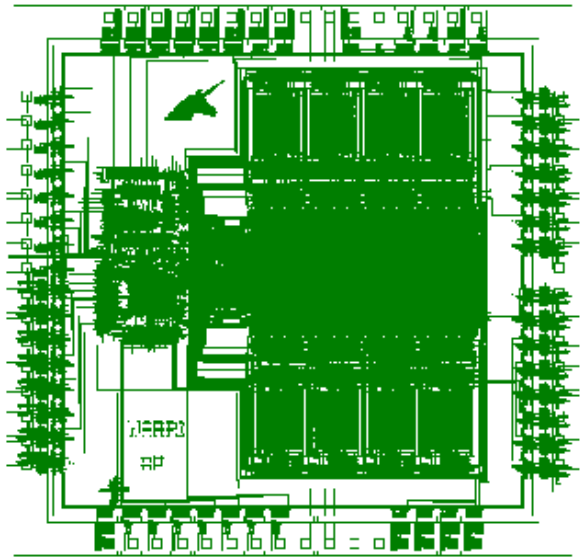
- I/O more limiting than computation $\rightarrow$ minimize storage.
- For $N \gg 1000000$, $\lg N$ is nontrivial $\rightarrow$ deprecate binary trees.
- Minimize explicit topology, expecially 3D.
- Plan for 3D; many 2D data structures not easily extensible to 3D, e.g., line sweep.
- E.g., Voronoi diagram: 2D is $\Theta(N \lg N)$. 3D is $\Theta\left(N^2\right)$
- Optimize function composition, e.g. Volume(Union(S)).

Uniform Grid

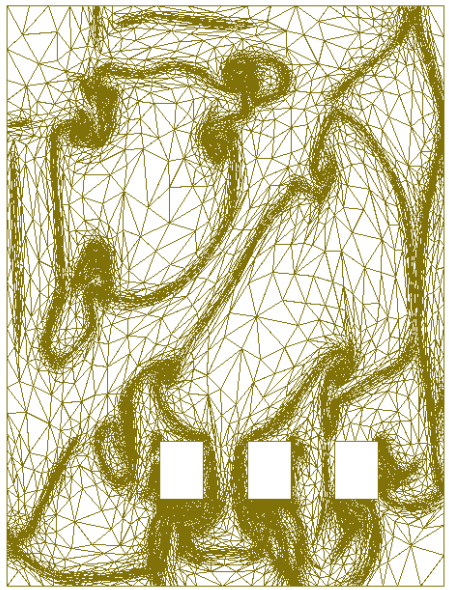
- Overlay a uniform 3D grid on the universe.
- For each input primitive — cube, face, edge — find overlapping cells.
- In each cell, store set of overlapping primitives.
- **Major disadvantage:** It's so simple that it apparently cannot work, especially for nonuniform data.
- **Major advantage:** For the operations I want to do (intersection, containment, etc), it works very well for any real data I've ever tried.



USGS Digital Line Graph



VLSI Design



Mesh

Uniform Grid Time Analysis

For i.i.d. edges (line segments), show that time to find edge–edge intersections in E^2 is linear in size(input+output) regardless of varying number of edges per cell.

- N edges, length 1/L, $G \times G$ grid.
- Expected # intersections $= \Theta\left(N^2 L^{-2}\right)$.
- Each edge overlaps $\leq 2(G/L+1)$ cells.
- $\eta \stackrel{\Delta}{=}$ # edges per cell, is Poisson distributed. $\overline{\eta} = \Theta(N/G^2(G/L+1))$.
- Expected total # intersection tests: $N^2/G^2(G/L+1)^2$.
- Total time: insert edges into cells + test for intersections. $T = \Theta\left(N(G/L+1) + N^2/G^2(G/L+1)^2\right)$.
- Minimized when $G = \Theta(L)$, giving $T = \Theta\left(N + N^2 L^{-2}\right)$.

Q.E.D.

Outline

Current grad students

- Wenli Li
- Salles Magalhães

Research Summary: Siting and ODETLAP

Wenli Li[1*], W. Randolph Franklin[1], Salles V. G. Magalhães[12], Marcus V. A. Andrade[2]

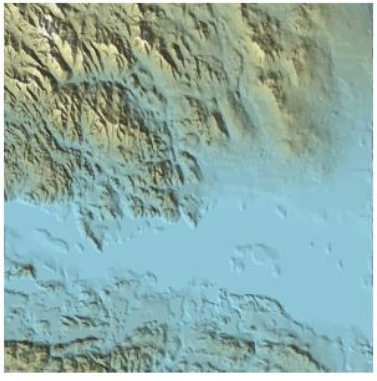
[1]Rensselaer Polytechnic Institute

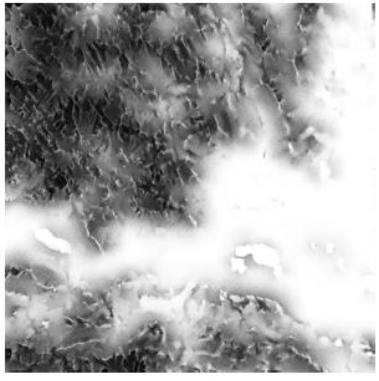
[2]Universidade Federal de Viçosa

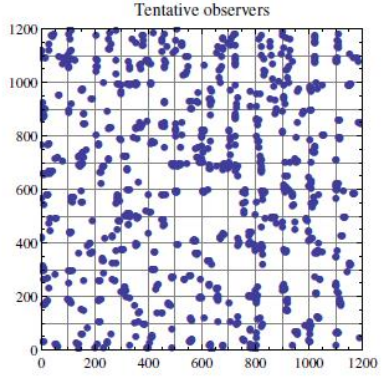
[*]liw9@rpi.edu

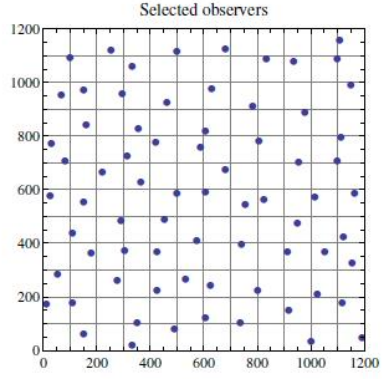
Multiple observer siting – Workflow

- Purpose: placing observers to cover targets on a terrain
- Workflow: *VIX* -> *FINDMAX* -> *VIEWSHED* -> *SITE*



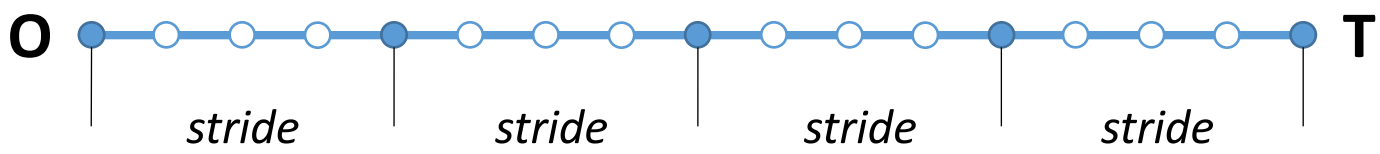






Multiple observer siting – Optimization 1

- *VIX*
  - Algorithm
    1. For each point
    2. Pick a number of random targets
    3. Compute the ratio of visible targets
  - Approximate visibility



  - Fixed *stride*: $1, 2, 4, 8, \ldots$
  - Increasing *stride*: 2^i

Multiple observer siting – Optimization 2

- *SITE*
  - Algorithm
    1. While not stop
    2. Compute the area of $V \cup C$ for the viewshed V of each unused observer
    3. Add the observer with the largest area
    4. Update the cumulative viewshed $C = C \cup V$
  - $Area(V \cup C)$: $O(n^2)$
  - $Area(V - C_V)$: $O(roi^2)$
  - Compute for unused observers within $2 \times roi$ of the last addition

Multiple observer siting – Parallelization

- OpenMP: compiler directives
- CUDA
  1. Compute visibility indices
  2. Select tentative observers
  3. Compute observer viewsheds
  4. Find observers within $2 \times roi$ of the last addition
  5. Compute the extra area of an observer viewshed
  6. Find the observer for addition
  7. Update the cumulative viewshed

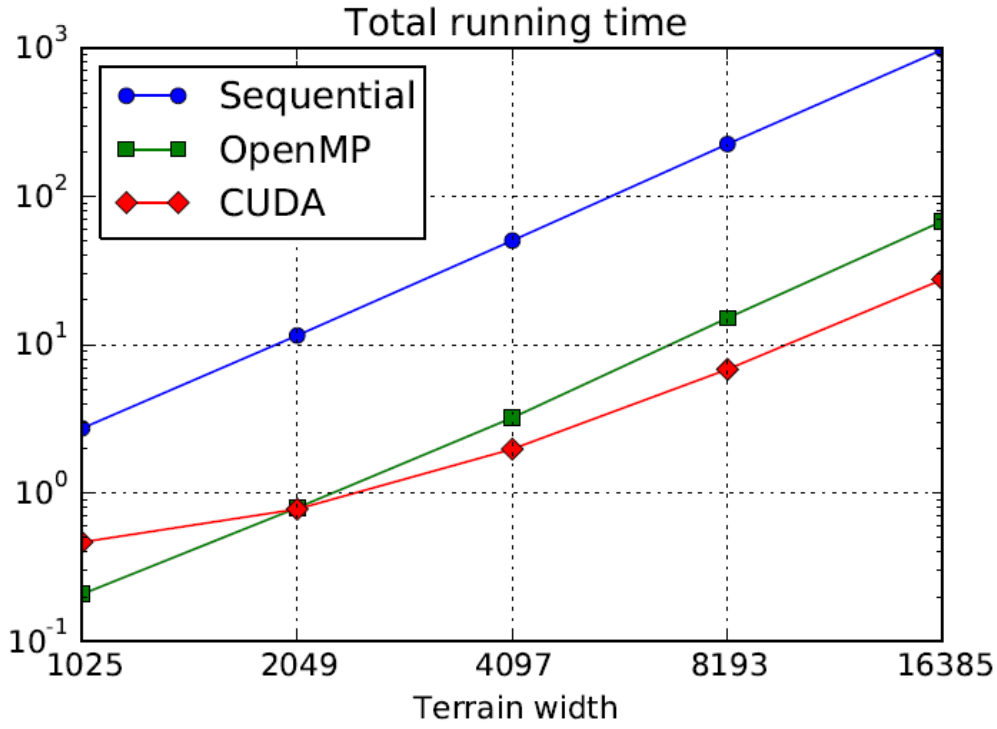
Multiple observer siting – Results 1

- 16K DEM, 26896 tentative observers
  - Running time of CUDA *VIX*
  - Percentage coverage
  - Number of selected observers

	30 targets			120 targets		
stride	Time (s)	Coverage	Observers	Time (s)	Coverage	Observers
1	87	95.5	25224	340	96.5	25108
2	50	95.5	25228	193	96.5	25130
4	30	95.5	25272	114	96.5	25137
8	19	95.5	25298	71	96.4	25214
2^i	11	96.0	25353	37	96.8	25239

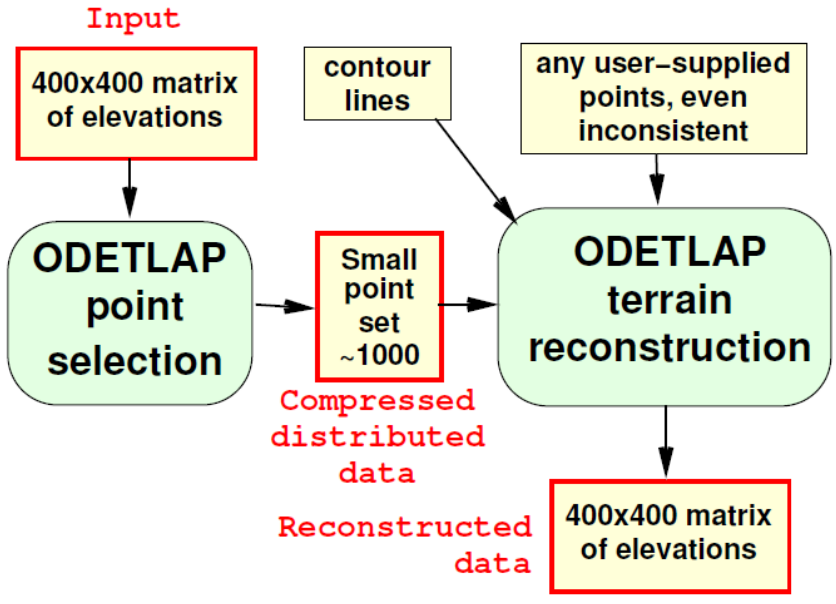
Multiple observer siting – Results 2

- Running time
  - Dataset: 1K, 2K, 4K, 8K, 16K DEMs
  - Hardware: two 8-core Xeon E5, Tesla K20Xm
- Speedup
  - OpenMP: 13—16
  - CUDA: 6—35



ODETLAP – Overview

- **Overdetermined** Laplacian Partial Differential Equations
- Two components: interpolation and lossy compression



(Franklin et al. CUDA-accelerated HDODETLAP: Lossy high dimensional gridded data compression. *Modern Accelerator Technologies for Geographic Information Science*, 2013.)

ODETLAP – Interpolation

- Two types of equations
  - Given a domain $m \times n$ and known points $\{(x_i, y_i, v_i)\}_k$
  - Averaging equation

$$4z(x,y) - z(x-1,y) - z(x+1,y) - z(x,y-1) - z(x,y+1) = 0$$

  - Smoothing factor R

$$4Rz(x,y) - Rz(\dots) - Rz(\dots) - Rz(\dots) - Rz(\dots) = 0$$

  - Known-value equation

$$z(x_i, y_i) = v_i$$

- Overdetermined system

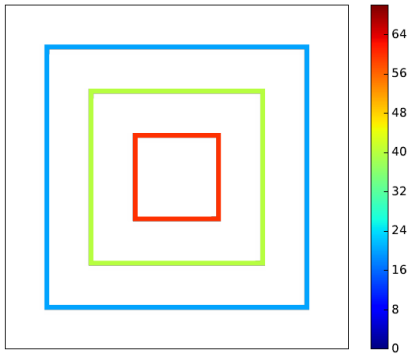
$$\boldsymbol{A}_{(mn+k)\times mn}\boldsymbol{x} = \boldsymbol{b}$$
$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$$

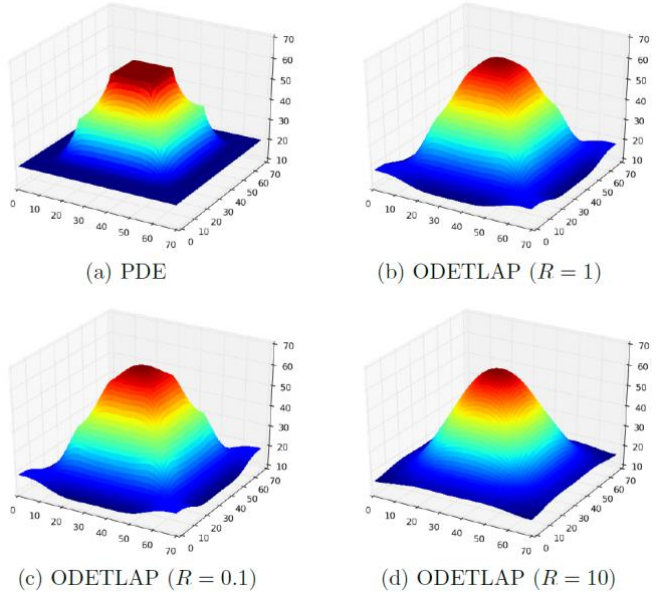
ODETLAP – Lossy compression

- Xie et al. Surface compression using over-determined Laplacian approximation. *SPIE 6697, Advanced Signal Processing Algorithms, Architectures, and Implementations XVII*, 2007.
- ODETLAP-based compression
  1. Select a set of initial points P using TIN construction
  2. Interpolate P using ODETLAP
  3. While stop condition is not satisfied
  4. Add a number of important points to P
  5. Interpolate P using ODETLAP

ODETLAP – Advantages over PDE

- Advantages
  - ODETLAP is overdetermined
  - ODETLAP can infer local extrema
  - Result is smoother across known points
  - R trades off accuracy vs. smoothness



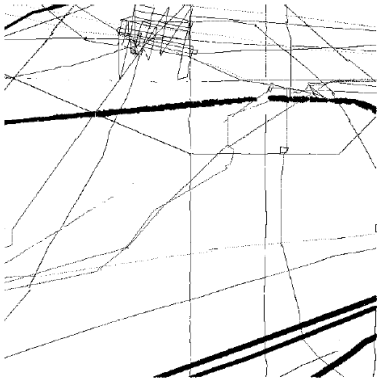


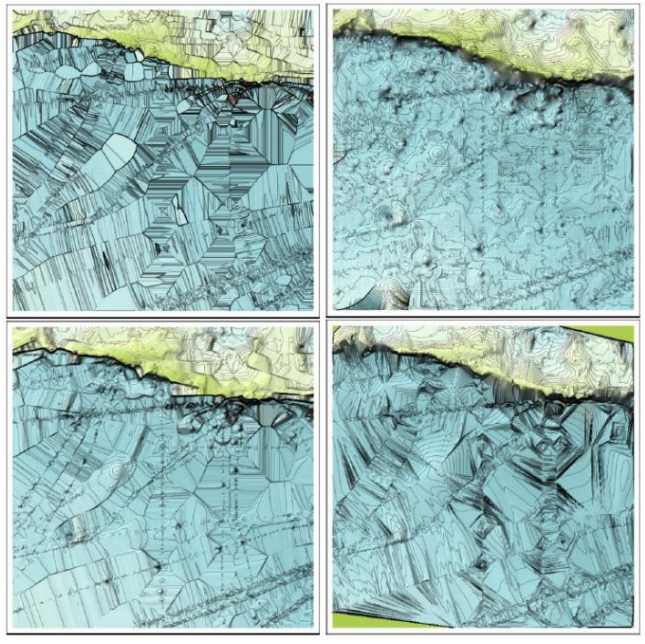
(a) PDE (b) ODETLAP ($R = 1$)

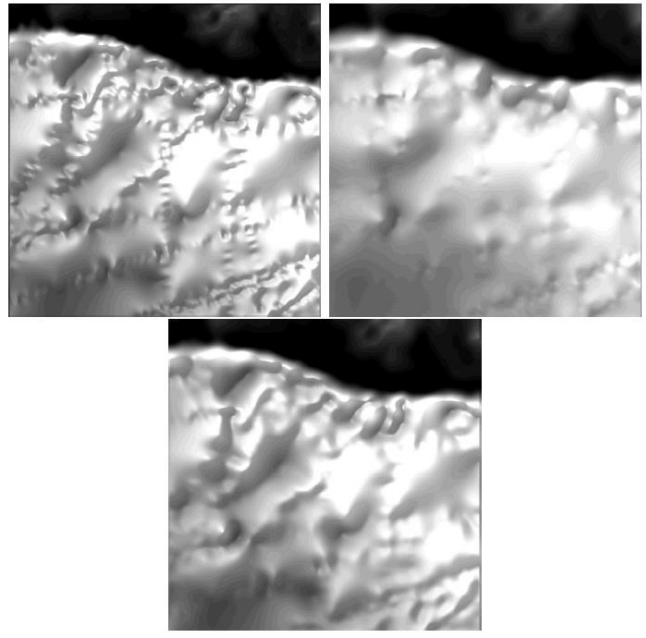
(c) ODETLAP ($R = 0.1$) (d) ODETLAP ($R = 10$)

ODETLAP – Application 1

- Lau et al. Sea floor bathymetry trackline surface fitting without visible artifacts using ODETLAP. *17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2009.
- Lau and Franklin. Automated artifact-free seafloor surface reconstruction with two-step ODETLAP. *SIGSPATIAL Special*, 2012.







Rensselaer Polytechnic Institute
Universidade Federal de Viçosa

RPI - Rensselaer Polytechnic Institute

Research topics in GIS

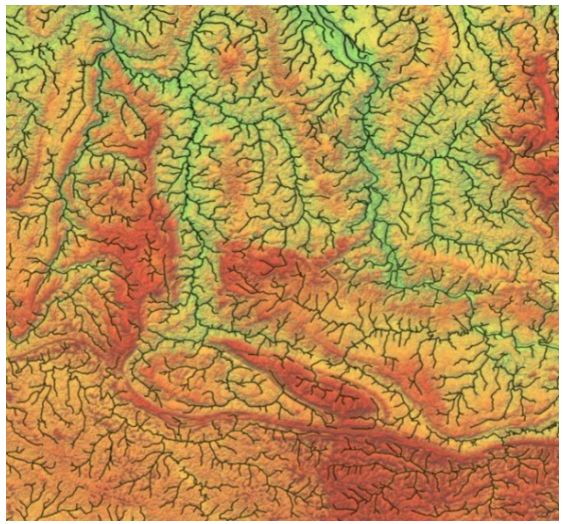
Marcus Andrade
Salles Magalhães
W. Randolph Franklin
Wenli Li

Our research

- Efficient parallel algorithms for GIS.
- Algorithms for raster and vector maps.
- Main fields in GIS:
  - Hydrography
  - Visibility
  - Operations with vector maps

Previous work: hydrography

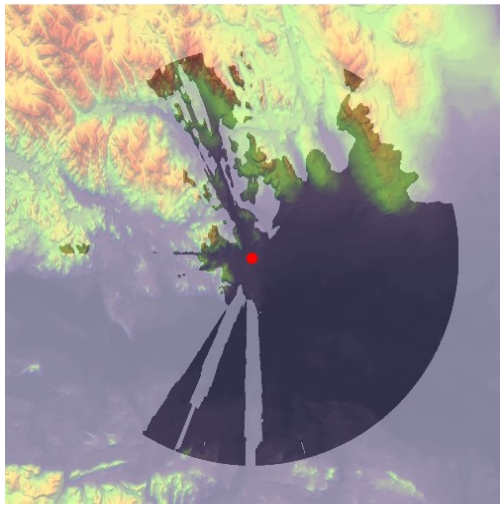
- RWFlood
  - Fast flow direction and accumulation
  - Linear-time algorithm
  - More than 100 times faster than others



- EMFlow
  - RWFlood for external memory
  - TiledMatrix (tiling+fast compression)
  - 20x faster than TerraFlow and r.watershed.seg

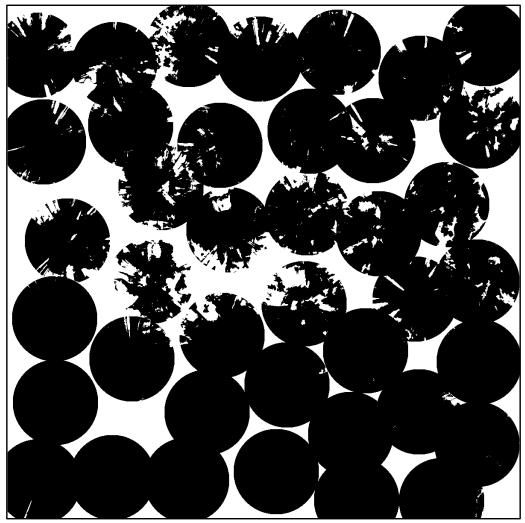
Previous work: visibility

- TiledVS
  - Visibility map computation on external memory
  - Uses TiledMatrix
- Parallel Viewshed
  - Multi-core implementation of the sweep-line viewshed
  - OpenMP
  - Up to 12x faster than the serial (using 16 threads)



Previous work: visibility

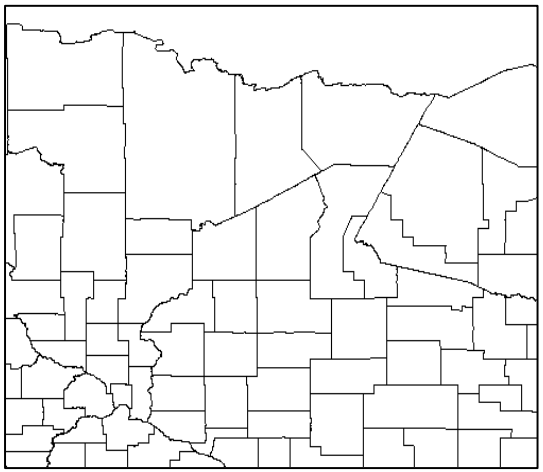
- GPU observer siting
  - Local search heuristic for observer siting
  - Given a solution S, iteractively replace S with its best neighbor
  - Neighbor(S): solution where an observer in S is replaced with an observer not in S.
  - Challenge: efficiently find the best neighbor
  - Solution: sparse matrices, adapted sparse-dense MM to compute visible areas.
    - Up to 3x faster than our previous GPU implementation.
    - Up to 7000x faster than our previous serial implementation (using dense matrices).

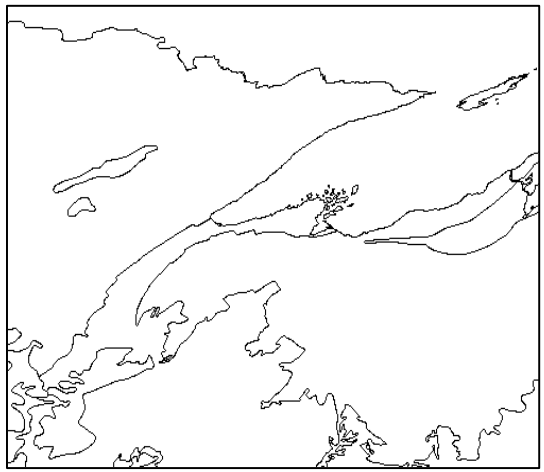


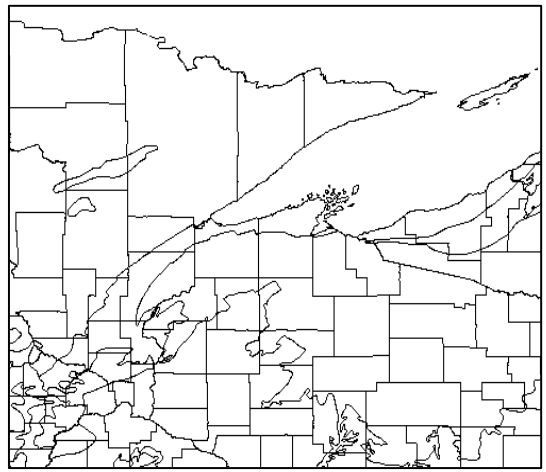
Current work: map intersection

- Finite precision of floating point → roundoff errors.
- Big amount of data → increase problem.

- Proposed solution: Rat-overlay
  - Uses rational numbers.
  - Parallelizable.

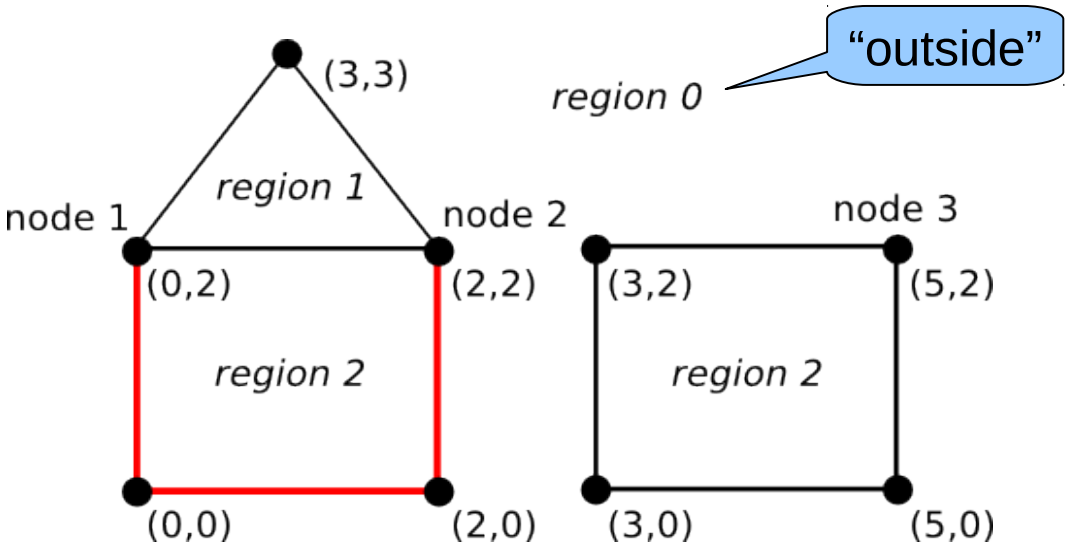






Current work: map intersection

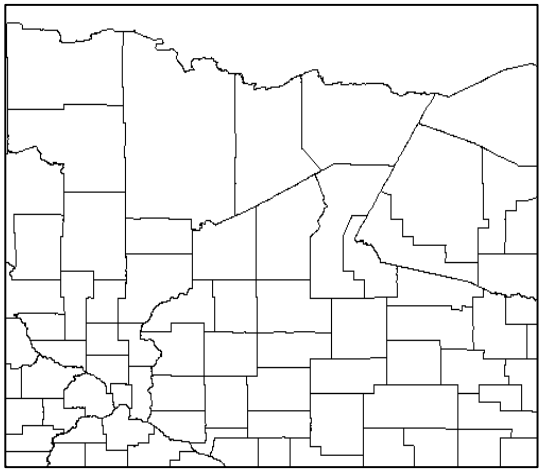
- Topological representation.
- Each region has one id.
- Edges represent boundaries.
- Sequence of edges bounding two regions:
  - chain: (id, #vertices, $node_0$, $node_1$, pol_{left}, pol_{right})

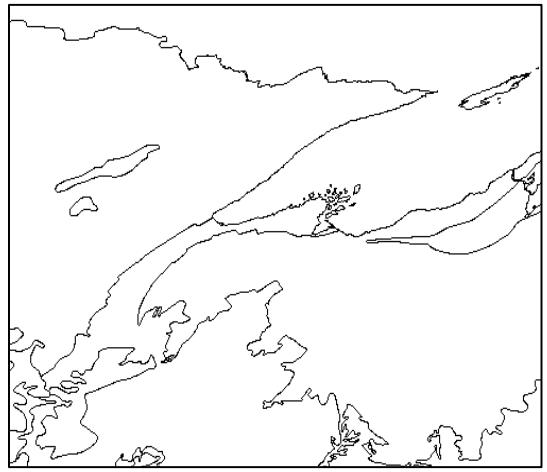


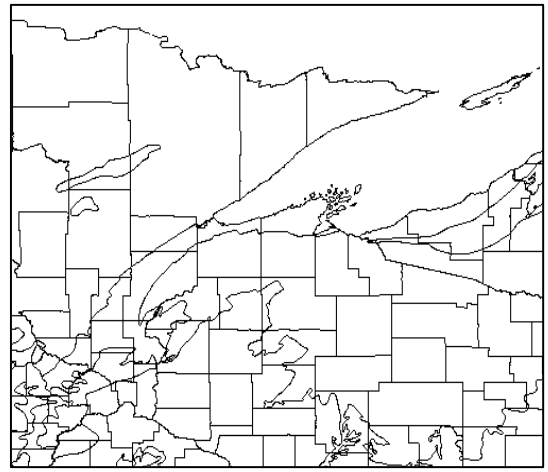
Chains:
(1,4,1,2,2,0)
(0,2);(0,0);(2,0);(2,2)
(2,2,1,2,1,2)
(0,2);(2,2)
(3,3,2,1,1,0)
(2,2);(3,3);(0,2)
(4,5,3,3,2,0)
(5,2);(3,2);(3,0);(5,0);(5,2)

Current work: map intersection

- Algorithm:
  - Find all intersections.
  - Locate vertices in the other map.
  - Compute output polygons.

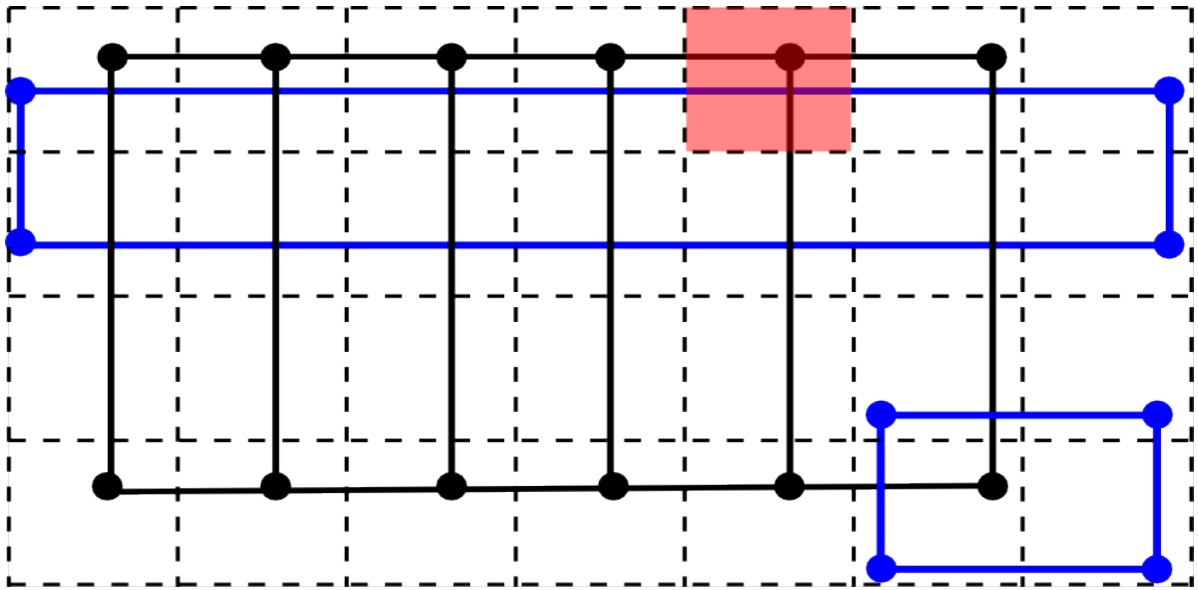






Current work: map intersection

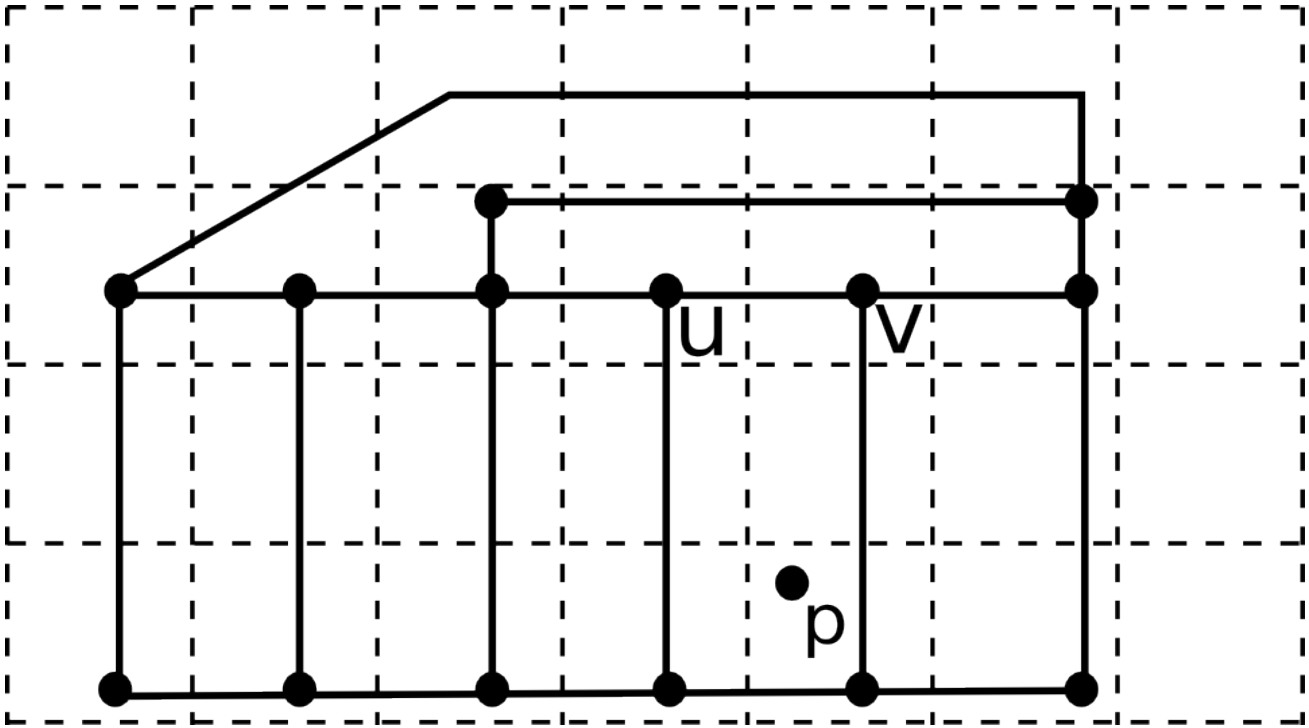
- Computing the intersections
- Test pair of edges for intersection.
- For efficiency: uniform grid.
  - Insert edges in grid cells (edge may be in several cells).
  - For each grid cell c, compute intersections in c.



4x7 uniform grid.
Blue map: 8 edges
Black map: 16 edges

Current work: map intersection

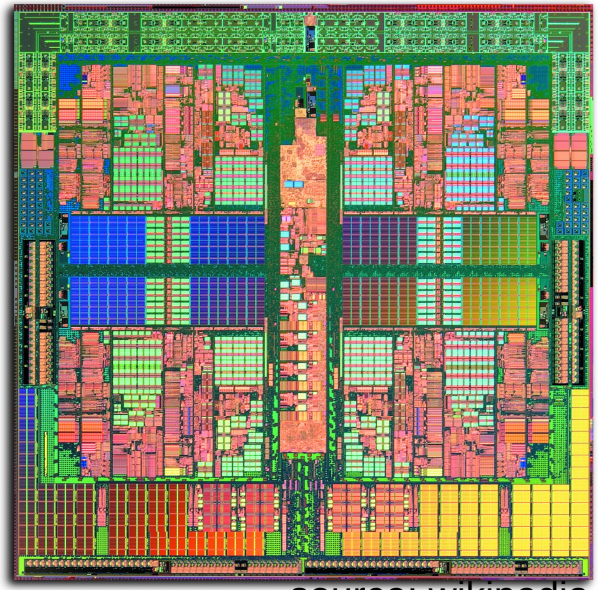
- Locating vertices in the other map
- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



Current work: map intersection

- This algorithm → few data dependency → very parallelizable.
  - Uniform grid creation: edges in parallel.
  - Locate vertices in polygons.
  - Compute intersections: cells in parallel.
  - Compute output edges: process input edges in parallel.

- Implemented using C++/OpenMP.



source: wikipedia

Current work: map intersection

- Computation is performed using rational numbers → no roundoff errors.
- Rat-overlay implemented using GMPXX.
- Special cases: simulation of simplicity.

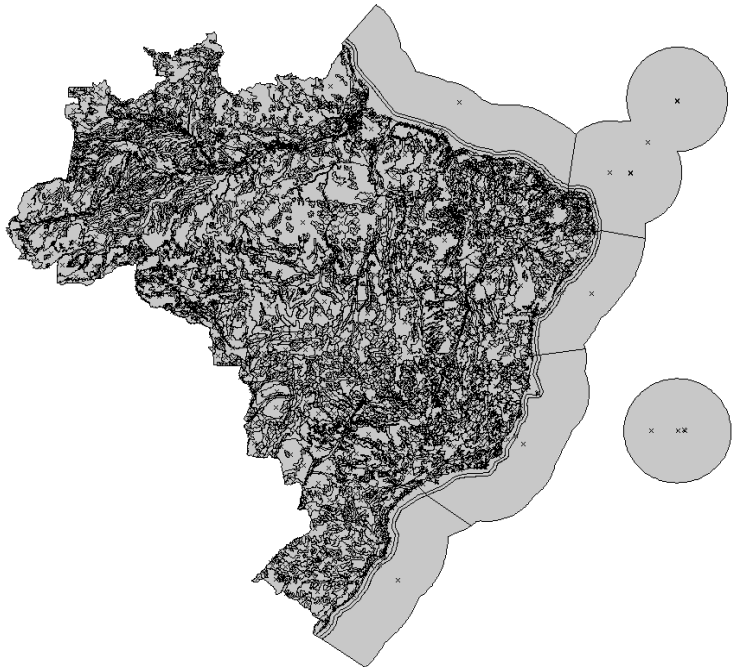
Current work: map intersection

- Rat-overlay implemented in C++ .
- Tests:
  - Dual Xeon E5-2687 → 16 cores / 32 threads.
  - 128 GiB of RAM.
  - Linux Mint 17

Current work: map intersection

- 2 Brazilian and 2 North American datasets.
- Shapefiles converted to our format.

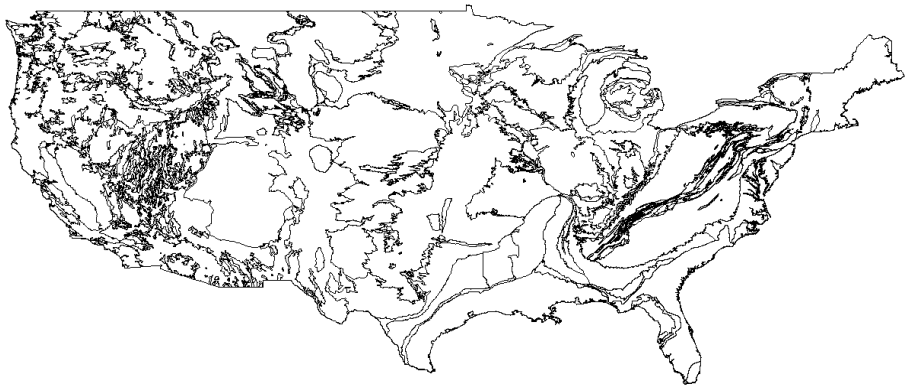
- BrCounty: 342,738 vertices, 2,959 polygons
- BrSoil: 258,961 vertices, 5,567 polygons.

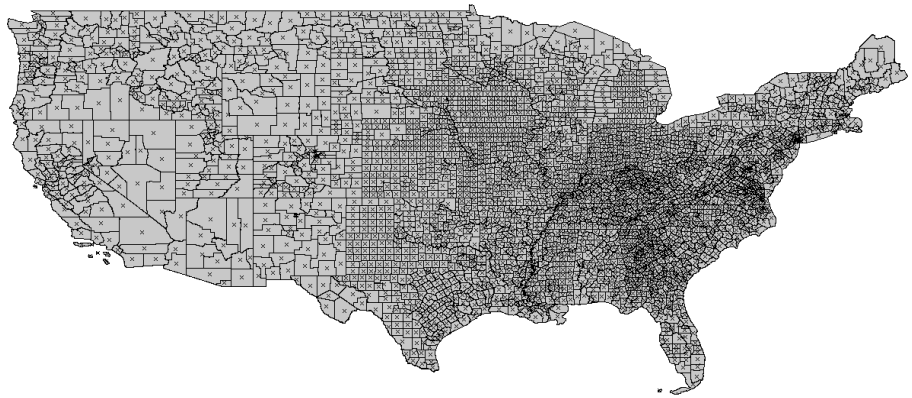


Current work: map intersection

- 2 Brazilian and 2 North American datasets.
- Shapefiles converted to our format.

- UsAquifers: 195,276 vertices, 3,552 polygons
- UsCounty: 3,648,726 vertices, 3,110 polygons





Current work: map intersection

- Sequential vs Parallel Rat-overlay vs GRASS GIS (sequential).
- Parallel:
  - Always faster than GRASS.
  - Speedup << 32
    - Critical sections.
    - 16 physical cores.
    - Amdahl's law.

Map 1	Map 2	# intersections	Grid size	Time (s)		
				Serial	Parallel	GRASS
BrCounty	BrCounty	105,754	2,000	34.5	11.5	30.3
BrSoil	BrSoil	56,246	2,000	23.3	7.4	32.3
BrCounty	BrSoil	20,860	1,000	16.1	5.9	81.7
UsAquifers	UsAquifers	50,329	8,000	37.2	11.9	47.3
UsCounty	UsCounty	300,511	16,000	625.5	124.4	175.0
UsCounty	UsAquifers	11,744	8,000	67.5	28.3	86.3

Outline

Future: Modeling of Valid Terrain

- What's the purpose of the HW and SW tools described earlier?
- My big long-term unsolved problem is to devise a mathematics of terrain.

Goals: Math that

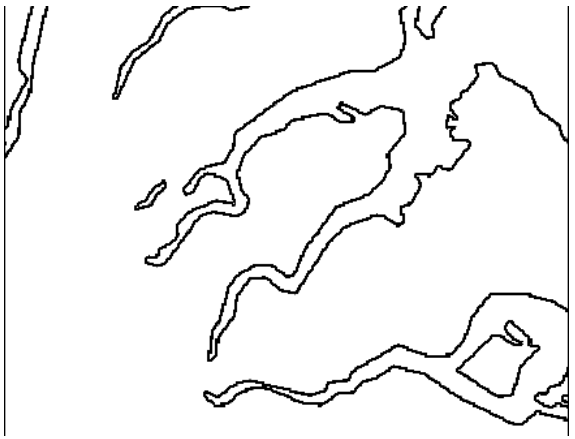
- allows the representation of only legal terrain (= height of land above geoid),
- minimizes what needs to be stated explicitly, and
- enforces global consistencies.

Why? To put compression and other ops on a logical foundation.

Terrain properties

Messy, not theoretically nice.

- Often discontinuous (C^{-1}).
- Many sharp local maxima.
- But very few local minima.
- Lateral symmetry breaking — major river systems.
- Different formation processes in different regions.



Peninsulas or fjords?

- Features do not superimpose linearly; two canyons cannot cross and add their elevations.
- C^∞ linear systems, e.g,. Fourier series, are wrong.
- Structure that people can recognize even though hard to formalize; see Figure.
- Multiple related layers (elevation, slope, hydrology).

Current representations

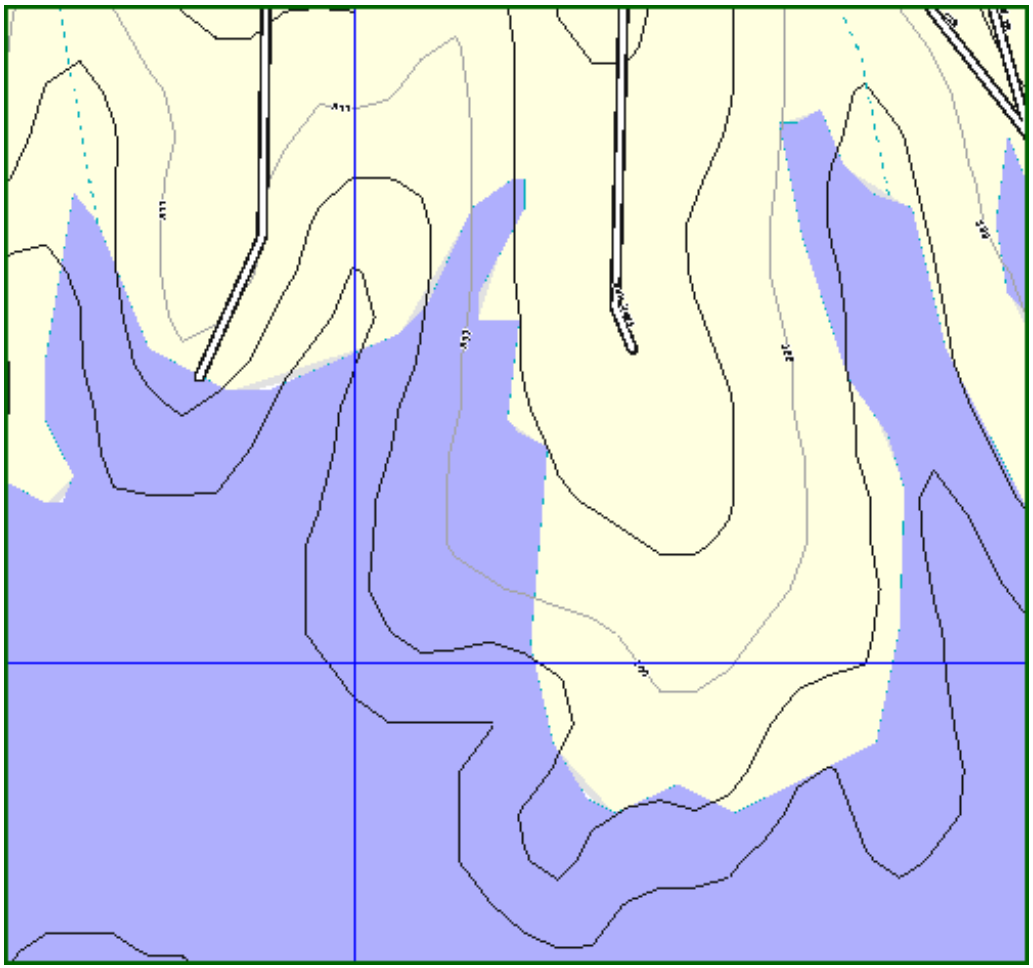
- Array of elevation posts.
- Triangular splines, linear or higher.
- Fourier series.
- Wavelets

Theory vs practice:

- Slope is derivative of elevation, but
- that amplifies errors, and
- lossy compression has errors, so
- maybe we want to store it explicitly.

Also, shoreline is a level set, but...

Inconsistencies between layers



Elevation contours crossing shoreline

Math should match physics

- Fourier series appropriate for small vibrations, not terrain.
- Truncating a series produces really bad terrain.
- Anything, like Morse complexes, assuming continuity is irrelevant.
- Fractal terrain is not terrain.
- Wavelets: how to enforce long-range consistency?
- Topology, by itself, is too weak.
- Terrain is not linear, not a sum of multiples of basis function.

Terrain formation by scooping

- Problem: Determine the appropriate operators, somewhere inside the range from conceptually shallow (ignoring all the geology) to deep (simulating every molecule).
- One solution: **Scooping.** Carve terrain from a block using a scoop that starts at some point, and following some trajectory, digs ever deeper until falling off the edge of the earth.
- Properties: Creates natural river systems w cliffs w/o local minima.
- Every sequence of scoops forms a legal terrain.
- Progressive transmission is easy.

Chris Stuetzle, *Representation and generation of terrain using mathematical modeling*, PhD, 2012.

Terrain formation by features

- Represent terrain as a sequence of features — hills, rivers, etc ..
- plus a combining rule.
- This matches how people describe terrain.
- Progressive transmission.
- The intelligence is in the combining rule.

How compact is this rep? How to evaluate it?

Implications of a better rep

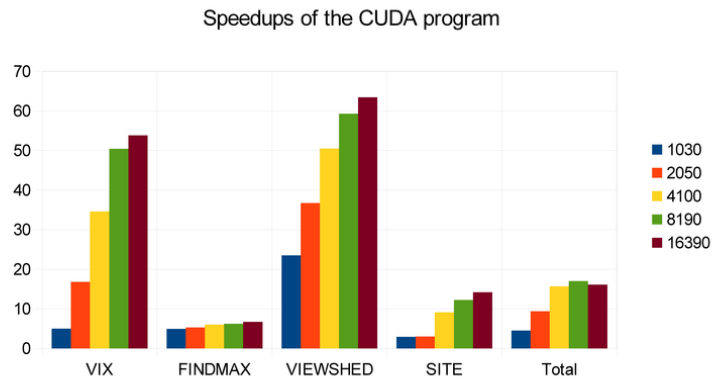
- Put earlier empirical work on a proper foundation.
- Formal analysis and design of compression.
- Maximum likelihood interpolation, w/o artifacts.
- Treat more sophisticated metrics, like suitability for operations like path planning, or recognizability.
- Close the loop to pre-computer descriptive geometry.

Outline Review

Summary

Guiding principles — to process big geometric and GIS datasets on parallel machines

- GPUs, memory are affordable.
- Build on powerful existing tools.
- Use minimal possible topology, and compact data structures.
- Use lots of memory; run BIG examples to show the linear time.



Source code (prototype quality) freely available for nonprofit research and education; I welcome stress tests and error reports.