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 km^2 [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

Advanced Signal Processing Algorithms, Architectures, and Implementations XVI, edited by Franklin T. Luk,
Proc. of SPIE Vol. 6313, 63130H, (2006) · 0277-786X/06/$15 · doi: 10.1117/12.684398

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 *2x2* 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).*

$$\begin{pmatrix} z1 \\ z2 \\ z3 \\ z4 \end{pmatrix} = \begin{pmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \\ x4 & y4 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

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 *2x2* kernel from the predefined origin. The *z* values are the elevations themselves and they change as the *2x2* 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.

	Horizontal Resolution	Minimum Elevation	Maximum Elevation
Dataset1	90m	278m	1591m
Dataset2	90m	122m	1076m
Dataset3	90m	29m	262m
Dataset4	30m	1097m	2014m
Dataset5	30m	1085m	1810m
Dataset6	30m	1398m	1627m

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.

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

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

	Number of Segments (Subset Size) after the Heuristic	Size of the Huffman Compressed Segment Bitmap	Size of the Plane Coefficients Table	Total	Compression Ratio
Dataset1	435	76,450 bytes	5,220 bytes	81,670 bytes	1:0.4537
Dataset2	262	67,316 bytes	3,144 bytes	70,460 bytes	1:0.3914
Dataset3	26	17,576 bytes	312 bytes	17,888 bytes	1:0.0994
Dataset4	172	60,248 bytes	2,064 bytes	62,312 bytes	1:0.3462
Dataset5	101	51,987bytes	1,212 bytes	53,199 bytes	1:0.2955
Dataset6	35	36,275 bytes	420 bytes	36,695 bytes	1:0.2039

Table. 4. Algorithm details for error bound of 20m. The original size of each of the datasets is 180,000 bytes.

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

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

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

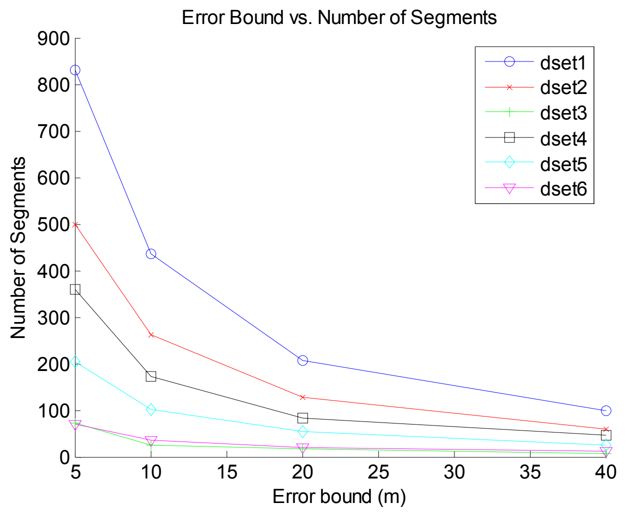

Fig. 1. With the increase in the error bound from 5m to 40m the number of segments decreases.

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

1. NC Floodplain Mapping Program, http://www.ncfloodmaps.com
2. USGS, Shuttle Radar Topography Mission, http://srtm.usgs.gov
3. W. R. Franklin, M. Inanc, Z. Xie, “Two novel surface representation techniques,” AutoCarto Vancouver, Washington 2006.
4. S. Osher, R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer-Verlag, 2002.
5. W. R. Franklin, A. Said, “Lossy compression of elevation data”, 7th International Symposium on Spatial Data Handling, 1996.
6. B. Turner, “Compressing Elevation Data,” http://www.vterrain.org/Elevation/index.html
7. MicroImages Inc., “JPEG2000: Lossy or Lossless?,” http://www.microimages.com/documentation/cplates/67jpeg2000LossyOrLossless.pdf
8. R. Matsuoka et al., “Quantitative analysis of image quality of lossy compression images,” ISPRS Istanbul, 2004.