Lossy Compression for Progressive Transmission of Digital Terrain Models

Marcus A. Andrade\(^1,2\), Zhongyi Xie\(^1\), W Randolph Franklin\(^1\), Barbara Cutler\(^1\), Metin Inanc\(^1\), Jonathan Muckell\(^1\), and Daniel M. Tracy\(^1\)

\(^1\) Rensselaer Polytechnic Institute, Troy, New York, 12180–3590, USA;
\(^2\) Universidade Federal de Viçosa, Viçosa, Brazil
{xiez,frankwr,inancm,muckej,tracyd}@rpi.edu
cutler@cs.rpi.edu,marcus@ecse.rpi.edu

Abstract. Key words: DEMs, Terrain Compression, Progressive Transmission, Levels of Details, Lossy Encoding.

1 Introduction

In the past 10 years the geographical information system (GIS) community has seen an explosion in the data volume and great improvement in the accuracy of data acquisition. High-resolution data at 10m is now widely available through USGS’ web site, which makes possible more realistic rendering of terrain features by all kinds of visualization softwares. Such growth of terrain data is reflected in the following aspects: Growth in digital spatial data and internet sites offering terrain data as freeware as United States Geological Survey’s 10 meter data sets. Also we see great growth in Mobile GIS services as 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 digitally demonstrate the earth’s surface and is a potential tool for terrain analysis at varied spatial and temporal scales.

Researchers over the world have developed lots of general-purpose algorithms that strive to maximize compression ratio while minimizing the processing time. However there are users who are more concerned about the quickness of the algorithm while also exists some users that expect the maximum compression ratio irrespective of the processing time due to the availability of supercomputers. In this paper we propose a DEM-specific lossy compression algorithm and its application in progressive transmission. Our algorithm is more concerned about the compression ratio than the and accuracy and CPU-time.

The following of this paper is organized as follows: Section 2 gives a brief summary of existing algorithms that does terrain compression, section 3 presents our algorithm for DEM compression. Section 5 describes in detail how we can progressively transmit the compressed data from ODETLAP. Section 6 summarizes the experiments we did and comparison with other methods.
2 Related Work

3 Over-determined Laplacian Approximation

3.1 Definition

The Over-determined Laplacian Approximation (ODETLAP) is an extension to Laplacian equation

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

The meaning of the equation is that for every non-border point identified by coordinate \((x, y)\) in our digital elevation matrix, the elevation \(z_{ij}\) is equal to the average of its neighbors. Handling of border points form a special case and is omitted due to the lack of deep theoretical interest. The idea of ODETLAP is when we have some but not complete 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. That is to add a new equation 2 for some points whose elevation is already known, while equation 1 is valid for all non-border points. During the approximation/interpolation process, we use a parameter \(R\) the value the relative importance of two sets of equations. Weighing equation 2 over equation 1 results in a more accurate surface which sacrifices smoothness. On the other hand, Weighing equation 1 over equation 2 gives us a smooth surface which is not very accurate. Because the desired reconstruction would conceal the contour lines which ODETLAP interpolates from, plus the input might be isolated points or irregular contour lines, we do need to value equation 1 somewhat. ODETLAP can be considered as a solver whose input is a set of known points \((x, y, z)\) and a interpolation parameter \(R\) and outputs the DEM matrix of the complete terrain.

\[ z_{ij} = h_{ij} \] (2)

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.

3.2 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%) , we use it as a decompressor in our algorithm. Figure 1 presents the flow chart of our algorithm. Our target DEM nowadays are mostly of \(400 \times 400\) resolution due to the processing capability and algorithm efficiency. The DEM firstly undergoes a points selection (See section 3.3) by external programs which pick a subset of posts \(S\) as input to ODETLAP solver currently implemented in Matlab. 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 an initial approximation of the elevation matrix. However, due to our pursuit of higher compression ratio, we normally pick a very small subset of points ($|S| = 1000$), consequently the initial approximation normally contains error that is above our needs for accuracy. So after obtaining the initial approximation, unless it accurate enough, (which happens when the original elevation matrix that 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, thus multiple points in the neighborhood is redundant, points added in the same step are checked against each other to avoid pairs that are within 5 pixels apart. The refinement steps end when overall RMS error falls below the required accuracy limit.

![Image of ODETLAP flowchart](image)

*Fig. 1. ODETLAP flowchart: Square box in Bisque are data and curved box in Skyblue are operations.*

### 3.3 Points Selection

We have tried several point selection methods, including regular grid, incremental TIN, visibility test, random selection and level set method. They vary quite a lot, in both way of working and efficiency. However, their results not quite much different, table ???. In all the methods that we tried, incremental TIN has the best support for progressive transmission, so we describe it in detail here.
Incremental TIN is based on the Franklin’s algorithm [?] which builds a triangulated irregular network (TIN) using a greedy insertion method to approximate a surface. Working in a breadth first search way, Incremental TIN iteratively splits the existing triangles by finding points that is farthest away. Points picked earlier are considered more important, so incremental TIN itself only reorders all the points by their importance, user can truncate the sequence and use only points that are most important. Because of this, our TIN based algorithm can be used to progressively transmit the compressed DEM data and receiver side can reconstruct the DEM using ODETLAP solver.

![Fig. 2. Triangulated Irregular Network](image)

### 3.4 Further Processing

Our ODETLAP based algorithm finally generates a subset of points from which ODETLAP solver could reconstruct a surface that is within error limit (usually RMS error no greater than 10 in our practice). The points can be compressed using text compressor like bzip2 or gzip. However, we can improve the compression ratio by extracting $x, y$ coordinates from $z$, process and compress them separately.

### 3.5 Run Length Encoding

$(x, y)$ coordinates are different from $z$ because they distribute evenly within the range $[1,400]$ 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. Because what we need to store here is only a binary bitmap
showing whether one point is existing in $S$ or not, there is no need to store the actual value. The method is summarized below:

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. $L \geq 766$, then use FFFFFFF as a two byte marker and use next two bytes for $L$.

We can see from the histogram of the run length (figure 3) 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 $400 \times 400$ regulation.

![Fig. 3. Histogram of two DEMs, we can see most runs are below 512.](image)

### 3.6 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 [2]. However, due to the processing overhead, only recently have such predictor been widely used [2].

The sequence $z$ that we are going to compress contains elevation information in the selected points by previous mentioned algorithm. Before processing, we order them by their corresponding $x, y$ coordinates so that during reconstruction process, the correlation can be reestablished. Linear predictors attempt to identify the redundancy in adjacent points.
There are several different ways to do linear prediction, and within JPEG standard, there are seven modes of prediction [?]. 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.

![Fig. 4. Compressing z values using linear prediction.](image)

The whole procedure is shown in figure 4. Given the point set $S$, we first sort them in the order of $x$ and $y$ coordinates. Then the $z$ sequence is extracted. We then compute the predicted sequence and correction sequence, which will be sent to some data compression software like bzip2.

The compressed size is given in table 1, the test cases are point sets from ODETLAP algorithm of size 1000. The test data used are three hilly data sets and three mountainous data sets, all of size 400 by 400. As we described in sections 3.6 and 3.5, the points are separately processing and compressed using linear prediction and run-length encoding. The entropies of our compression and compressed size using purely bzip2 is also listed in the table for further reference.

<table>
<thead>
<tr>
<th></th>
<th>Hill1 compr. xy</th>
<th>Hill2 compr. xy</th>
<th>Hill3 compr. xy</th>
<th>Mtn1 compr. z</th>
<th>Mtn2 compr. z</th>
<th>Mtn3 compr. z</th>
</tr>
</thead>
<tbody>
<tr>
<td>compr. xy size</td>
<td>1250B</td>
<td>1243B</td>
<td>1279B</td>
<td>1228B</td>
<td>1244B</td>
<td>1241B</td>
</tr>
<tr>
<td>compr. z size</td>
<td>1304B</td>
<td>1354B</td>
<td>1200B</td>
<td>1456B</td>
<td>1424B</td>
<td>1503B</td>
</tr>
<tr>
<td>Total compr.size</td>
<td>2554B</td>
<td>2597B</td>
<td>2488B</td>
<td>2684B</td>
<td>2668B</td>
<td>2744B</td>
</tr>
<tr>
<td>Entropy (b.p.e.)</td>
<td>2.554</td>
<td>2.597</td>
<td>2.488</td>
<td>2.684</td>
<td>2.668</td>
<td>2.744</td>
</tr>
<tr>
<td>Bzip2 size</td>
<td>4136B</td>
<td>4234B</td>
<td>4025B</td>
<td>4328B</td>
<td>4416B</td>
<td>4355B</td>
</tr>
</tbody>
</table>

Table 1. Compression of 1000 points: Split into $xy$ bitmap and $z$ sequence, then use run-length encoding on $xy$ and linear prediction on $z$. Entropies (in bit per elevation is also listed)
4 ODETLAP based Progressive Transmission

5 Result and Analysis

<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>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>Compr. 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># 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>8.49</td>
<td>9.93</td>
<td>8.31</td>
<td>9.48</td>
<td>9.55</td>
<td>9.68</td>
</tr>
<tr>
<td>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>

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.