Progressive Transmission of Surfaces with Geometric Constraints

by

L. Scott Johnson

Bachelor of Science
University of South Carolina, 1991

Submitted in Partial Fulfillment of the

Requirements for the Degree of Master of Science in the

Department of Mathematics

University of South Carolina

2004

Department of Mathematics Director of Thesis	Department of Mathematics Second Reader

Dean of the Graduate School

Acknowledgements

I would like to thank Dr. Sharpley for his enlightening lectures in my undergraduate years, for remembering me years later to invite me to work in the IMI, for his tutelage and guidance during my decade with the IMI, for motivating me to get into the graduate program, and for his constructive comments in reviewing my thesis.

I would also like to thank Dr. DeVore for his ideas, guidance, and honest appraisals of my efforts with the IMI.

I also thank Dr. Lane for his help tackling the topics I've presented to him these past years and for his insights on and critiques of my thesis.

I must also thank my family, first Jennifer for her support and devotion all these years, and for believing in me, and also Aden and Danielle for providing a guilt-free reason to avoid my studies and for letting me study in peace on the rare occasions that I asked.

And finally, I thank my mom and dad for getting me off to such a great start, and my brother for his pacing along the way.

Abstract

This paper describes a method of decomposing and progressively transmitting elevation data (height maps) in a highly compressed form that preserves its geometric nature. This is important in many critical military and civilian applications such as autonomous navigation of ground and aerial vehicles, surface hydrology, emergency response, and route planning. The main steps in this method are representing the elevation data as a set of skeletal curves (contour lines as well as ridges and ravines) and the use of multiresolution analysis to represent those curves. Used together, these steps allow the data to be sent progressively (a coarse approximation with little information which is gradually refined by sending more and more information) in a client-server setting. These steps are followed by interpolation methods for fill-in on the client side.

Table of Contents

1. Surfaces as Image Data.......... 1

Multiresolution Analysis of Images 1

Progressive Transmission 4

Improvements for Surfaces 8

2. Server (Data Decomposition).......... 9

Extract Critical Points.......... 10

Extract Level Curves 14

Extract Ridge Curves and Ravine Curves.......... 17

Schedule Sections for Encoding.......... 20

Decompose Curves 21

3. Communication (Data Encoding and Decoding) 26

Progressive Transmission 26

4. Client (Data Reconstruction) 29

Reconstruct Approximated Curves 29

Create Piecewise Linear Triangulation.......... 29

5. Results.......... 31

Bibliography 35

Appendix: Annotated Bitstream Example 37

List of Figures

Figure 1: Discrete Wavelet Transform for an Image 2

Figure 2: Quad Tree Partitioning of an Image 3

Figure 3: Client-Server Model 4

Figure 4: Bitstream Encoding of a Minimal Tree 6

Figure 5: Coefficients Decomposed Into Bitplanes 7

Figure 6: Triangulation with Type 1 Diagonals 11

Figure 7: Marching Squares Algorithm 16

Figure 8: Morse Structure of Level Curves 17

Figure 9: Sample Terrain Data and Corresponding Flow Graph. 18

Figure 10: Sample Flow Accumulation and Resulting Ravine. 19

Figure 11: Multiresolution Analysis of a Curve 23

Figure 12: Results 32

Figure 13: Maple Mountain Dataset, Selected Curves, and Initial Approximation 33

Figure 14: Maple Mountain Reconstruction through Next 3 Bitplanes (2, 1, and 0) 34

1. Surfaces as Image Data

The elevation data used in our research comes in a uniform rectangular grid of height values expressed in meters. The data points have a uniform distance between adjacent values, which is typically the same in both the X and Y directions. In this respect, terrain data has a natural resemblance to data for grayscale images, but it contains inherent geometry not present in images. However, with all of the resources that have been devoted to image compression, applying the same techniques for elevation data is an easy first attempt for this task, and it helps illustrate some of the concepts.

In the following sections, the basics of multiresolution analysis and progressive transmission of images using wavelets are described.

Multiresolution Analysis of Images

For multiresolution analysis (MRA), a grayscale image (or a height map) is treated as a piecewise continuous real-valued function of two variables. The value of the function is only known at discrete (and uniformly spaced) sample points (the pixels of the image or the postings of the height map), and must be interpolated by an appropriate method to produce a full image. Usually this is implemented through a triangulation for terrain, or by piecewise constant fill (pixels) for images.

This pixel representation of the image is converted into a wavelet representation using a Discrete Wavelet Transform (DWT). Each row of pixels is treated as a 1-D signal (*i.e.*, a sampling of a continuous function of one variable), and the DWT is applied to that signal.

This produces an ordered set of "coarse" coefficients (C) and an ordered set of "detail" coefficients (D), with the number of coefficients in each of these ordered sets equal to half the number of sampled values (pixels) in the signal at the finest resolution, or the highest level.

The resulting coefficients (from all rows) are then processed again by applying the DWT to the columns instead of the rows.

For a 2^j by 2^j image, this produces four 2^{j-1} by 2^{j-1} sets of coefficients (one set of the scale coefficient for the columns of the scale coefficient of the rows, one set of the scale coefficients for the columns of the detail coefficients of the rows, and so on). We will denote these sets CC, CD, DC, and DD, respectively. We iterate this process on the CC sets (each time replacing the CC set with four new, smaller sets) until the CC set is a singleton coefficient. See Figure 1 for an example of the first two iterations of this process.

Figure 1: Discrete Wavelet Transform for an Image

This rearrangement of data is lossless. The original data can be completely recovered by applying the Inverse Discrete Wavelet Transform (IDWT). The IDWT is applied beginning with the singleton coefficients, first to build the columns (with just 2 elements on the first iteration) then the rows (to get a 2×2 square of elements), and iterating until the original image is restored.

These coefficients of the DWT are then rearranged into four quad trees, one for each group of coefficients (CC, CD, DC, and DD). The trees are ordered in increasing detail from top to bottom, with the coarsest level coefficient at the root and the finest level of detail at the leaves. The single remaining CC coefficient is the root and sole element of its tree. For each of the other groups (CD, DC, DD), the singleton coefficient produced in the final iteration is treated as the root node of the respective tree. Each root has four children: the upper left, upper right, lower left, and lower right coefficients of the previous iteration (one level finer detail). Each of those four has the four coefficients from the same spatial position of the next finer level of detail as its children and so on.

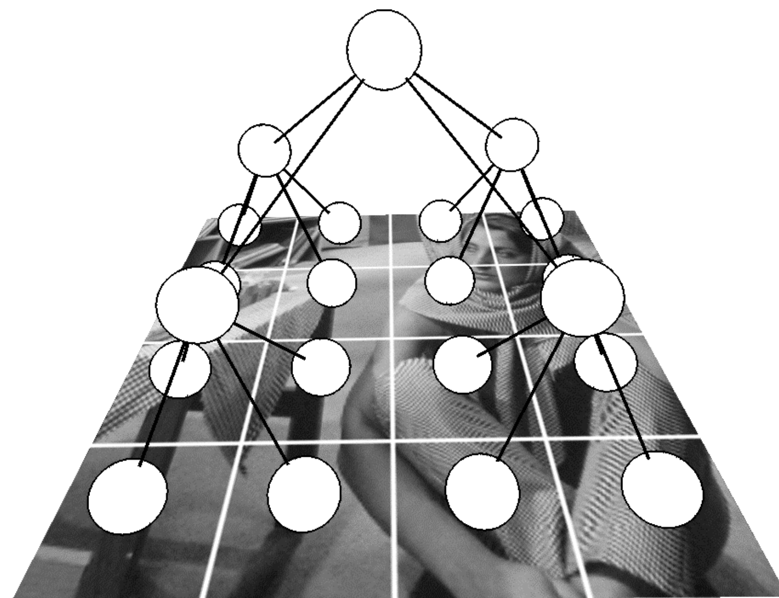


Figure 2: Quad Tree Partitioning of an Image

Progressive Transmission

In real-world applications, the end user of the data (the client) may not have enough local (on-board) storage to retrieve and store all of the data that he or she would need, so the client has to rely on a remote server (one with enough storage capacity to store all the necessary data) to transmit the portions of the data that he or she needs as it is needed. In the field, the rate of data communication may not be high enough to send the all of the needed data quickly enough to be useful. A solution is to send a very small portion of the data first, but enough for the client to reconstruct part of the data, and then send another portion of the data. This is progressive transmission. Internet users see this sort of progressive transmission routinely when visiting pages with large images that have been properly encoded, typically with three levels of detail.

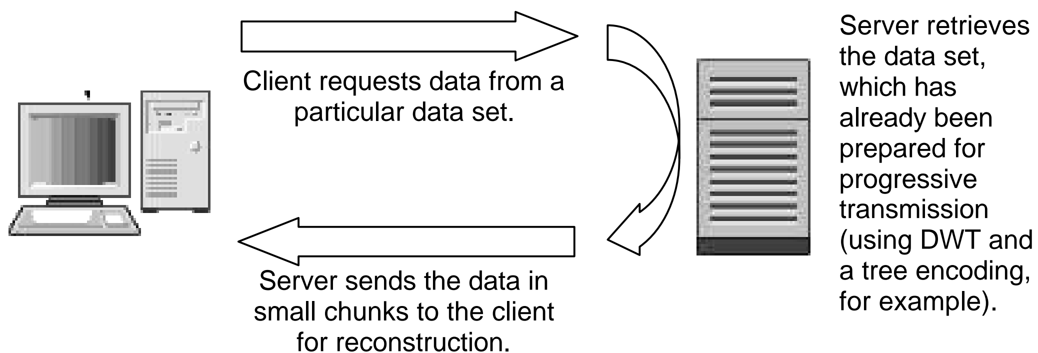


Figure 3: Client-Server Model

For progressive transmission from a server to client as described briefly above, the image data is broken up into a sequence of pieces. The first piece contains enough information to reconstruct a coarse approximation to the (full) image. The next piece contains additional information that can be used to create a more detailed approximation of the image when combined with the first piece. And so on.

The rearrangement of the original image data into the DWT coefficients lends itself to (among other things) compression by discarding small coefficients (*i.e.*, thresholding) and reducing the precision required of larger coefficients (*i.e.*, quantization). In (Cohen, *et al.*, [2]), it is proved that storage of just the reduced-precision large coefficients and their positions in the trees (and treating the other coefficients as zero) produces optimal quality reconstruction for the compression achieved, asymptotically (to within constants). This is framed in terms of Kolmogorov entropy and can be motivated in terms of the total boundedness of compact sets, ε-nets, and code books; see [2] for details. The coefficients at different levels of the quad tree can be weighted differently for quantization depending on which L_p space is being used for the error metric. For images, the L_2 metric is usually used, since it is a good approximation for the "eye metric" (that is, the more different two things look to the eye, the bigger the L_2 distance between them tends to be).

For progressive transmission, first a highly compressed, low quality image is sent (sending just a small number of coefficients with low precision). Then more levels of detail are sent by transmitting the positions in the tree of new coefficients and additional precision for the coefficients previously sent. The progressive algorithm trades off some compression for the ability to send the data progressively. Instead of just sending the positions of the coefficients as a unit, extra information is needed (more bits must be transmitted) in the progressive scheme to send the preliminary trees (the positions of the preliminary coefficients) first and then extend that tree to reflect the positions of the ultimate coefficients for a given level of reconstruction. Both methods use the same amount of information for the values of the coefficients.

Quantization and thresholding are both applied by selecting a bitplane (*i.e.*, by selecting an integer j and using 2^j as both the threshold level and the quantization level). To begin, we take the largest coefficient (in magnitude) to find the first (most significant) bitplane needed for the dataset. This is the bitplane used for the first level in the progressive set of levels. Each subsequent level uses the next lower power of two, down to some pre-determined level (the finest level of resolution needed for the application).

For the first bitplane level, the positions of the non-zero coefficients are sent by growing each tree from the root. We use the minimal tree necessary to hold all of the non-zero coefficients, even though some of the (non-leaf node) coefficients in the tree may be zero. The root node is always needed, since we can assume that there is always at least one non-zero coefficient somewhere in the tree. If all the coefficients were zero, then the reconstructed image would just be a constant gray (or otherwise uninteresting picture, depending on the type of wavelet used). For each node already known to be in the tree (each of the four roots, to start), four bits are sent indicating which of the four child nodes are in the tree. The tree is sent recursively in depth-first order, meaning each child is handled immediately after the 1 bit is sent indicating its presence. See Figure 4 for an example of how this encoding would proceed for a binary tree (the extension to a quad tree is trivial).

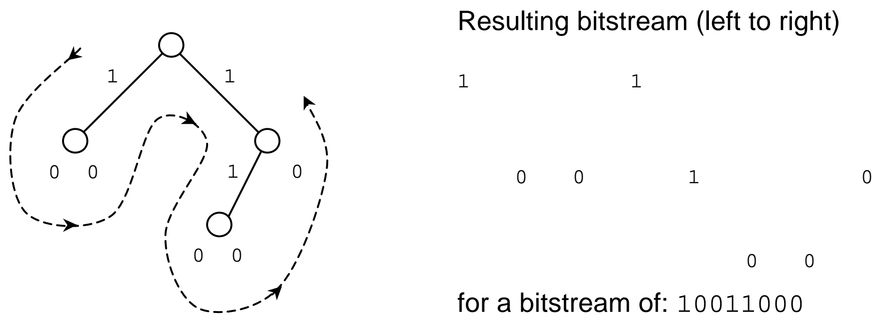


Figure 4: Bitstream Encoding of a Minimal Tree

For each subsequent level, we send only the positions of the non-zero coefficient that were not in the previous level. Each level's minimal tree contains the previous level's tree, so we grow them from the previous tree instead of starting over at the root each time.

Since these growth bits are mostly zeros, a lossless arithmetic encoder like Q-Coder (Pennebaker, *et al.*, [11]) can be used to achieve higher compression. Additional adjustments can be used as well. For example, the maximum depth of each tree could be sent at the start to avoid later sending bits to indicate that none of the four children of the nodes at that depth are present.

Once the positions are transmitted, the bits for the coefficients are sent. For each coefficient in the trees, one bit is sent to update the magnitude. This bit (0 or 1) is multiplied by the current bitplane value 2^j and added to the previous (quantized) value of the coefficient (see Figure 5). For new nodes (node that were not present in the previous level's tree), this value sets the magnitude of the coefficient. Note that the bit (and therefore the value) may still be zero for nodes that are needed for connectivity. For new leaf nodes (node with no children in the current tree, not necessarily nodes at the maximum depth of the tree), the bit must be 1, and so that bit is not transmitted.

Finally, for each coefficient in new nodes (even the interior ones with zero magnitude at the current quantization level), a sign bit is sent.

	12.5	-2.0	-0.5
Sign	0	1	1
BP 3	1	0	0
BP 2	1	0	0
BP 1	0	1	0
BP 0	0	0	0
	.	.	.
BP -1	1	0	1

Figure 5: Coefficients Decomposed Into Bitplanes

Neither the string of sign bits nor the string of coefficient update bits benefit from additional compression techniques, since they are effectively random and uniformly distributed, but it is usually more convenient to run just one Q-Coder and encode all of the bits.

Improvements for Surfaces

Treating the surface data as an image is somewhat effective, but improvements in both geometric quality and efficiency of processing are required for typical usages of terrain data. Image data is typically piecewise smooth, but terrain data is continuous, reflects inherent geometry formed historically through geologic processes, and are somewhat smoother. The desired properties of the reconstruction are different as well. To get a “good picture” in image processing, the error is usually minimized in the L_2 sense. The L_∞ metric, however, is better suited to producing a “good” reconstruction of the terrain, especially where local extrema are concerned. The Hausdorff metric is even better suited for terrain, but is computationally more expensive to perform. As mentioned earlier, the coefficients of the DWT can be scaled (based on their level in the multiresolution ladder) to minimize the L_∞ error instead of minimizing the L_2 error for a given degree of quantization, but that is a dataset-wide operation (and does not focus particularly on the local minima and maxima). The following sections describe a method of selecting and compressing skeletal curves (parameterized curves in threespace) from which to reconstruct the terrain.

2. Server (Data Decomposition)

Instead of using the pixel/posting data directly for the multiresolution analysis, the server first finds a set of curves to serve as a “skeleton” of the terrain. It is on these curves that the server performs the multiresolution analysis. The server selects two types of curves: level curves and ridge/ravine curves.

Level curves and contour maps provide a natural way of understanding terrain data (as many hikers already know). Capturing the locations of the critical points (local minima, local maxima, and saddle points) along with the level curves that pass through these points are helpful in understanding the basic topography of any terrain. These curves define the Morse structure of a given plot of terrain and provide local information about the monotone regions that comprise the terrain. A monotone increasing (decreasing) region is a contiguous patch P of the surface with an “outside” boundary B_O and either a contiguous “inside” boundary B_I or an interior point designated as the inside border B_I such that, for every point p in P, there exists an increasing (decreasing) curve in P from a point in B_O to a point in B_I that passes through p.

Other key features to understanding the shape and characteristics of a landscape are the ravines and ridges that cut through the terrain. Ravines and ridges are generally curves that follow paths of high curvature. By introducing a water flow model to the terrain information, one can loosely define what a ravine is by its natural characteristic of being a local accumulator of water or a place where water flow convergences. In an opposite role, ridges can be defined as locations of diverging water flow. Given a terrain, ridges

have the characteristic of being essentially at the locations of where ravines would be if the terrain data were inverted. With this relationship, ridges and ravines are treated numerically the same. With a sufficient number of these different curves, an accurate understanding of the terrain can be gained and then be implemented in generating good and efficient approximations to the original data set.

To prepare a dataset for progressive transmission, the server must first choose a set of curves to represent the geometry of the data. The level curves are chosen by identifying the critical points in the dataset (local extrema and saddle points) to partition the dataset into maximal monotone sections. The boundaries of those sections are the level curves we use. The ravine curves are chosen by identifying areas of high inflow from the nearby neighborhood. Ridges are chosen by looking for ravines in the z-inverted dataset. The following sections describe each of these steps (finding critical points, extracting level curves, and extracting ridges and ravines) in greater detail.

Extract Critical Points

In our considerations we can use either 6- or 8-neighborhood stencils around each posting. The 8-neighborhood of a posting consists of all postings adjacent to it, orthogonally or diagonally. This formulation has the benefit of treating all directions equally. The 6-neighborhood that we use is the 8-neighborhood with the northwest and southeast corners excluded. This seems more a natural choice for terrain compression since it corresponds to the regular, type I triangulation (see Figure 6) which underlies the piecewise linear functions used for visualization of the uncompressed terrain data. For concreteness in our exposition we will assume 6-neighborhood. Thus two postings are

neighbors if and only if they are in the 6-neighborhood of one another. Using the case of 8-neighborhood requires only minor modifications

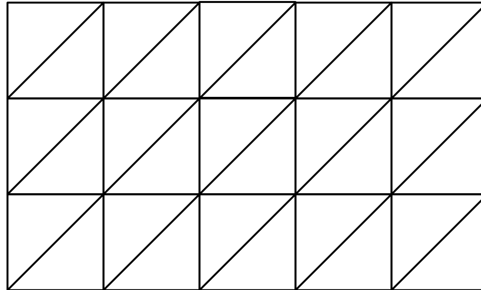
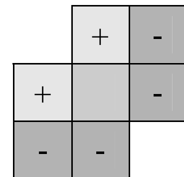


Figure 6: Triangulation with Type 1 Diagonals

To extract the critical points of a dataset, one needs only to examine the adjacent postings. This is a straightforward process if adjacent postings always have different values. If all neighbors of a given posting have greater (smaller) height then the posting is a local minimum (maximum). Further, if the neighborhood (hexagonal polygon) of a posting consists of two contiguous "wedges" (*i.e.*, groupings of neighbors), one of higher and one of lower postings (see example at right), then the posting is a regular point, and if the neighborhood of a posting consists of four or more alternating wedges of higher and lower postings, then the posting is a saddle point. When a posting has a neighbor of the same height, however, the boundary of the same height neighborhood must be examined as a whole (treating the region of same height postings as a single posting to determine if it is contains a critical point). This has a few additional levels of complexity over the single posting method. The obvious source of complexity is that an arbitrary number of neighbors (or neighborhoods) must now be checked. Additionally, it could be that the same height neighborhood has an



interior boundary in addition to its exterior boundary. This latter case requires additional clarification of the properties that identify a region as a particular type of critical point.

As a preliminary step, we apply the simple recipe from above to every posting with the agreement that if a posting has a same height neighbor then that posting is left undecided. Thus, with one scan through the data, we classify each posting as a local minimum, local maximum, saddle, regular, or undecided point. This preliminary step is optional in the sense that omitting the step would not affect the final results. This step excludes the one-posting connected iso-components from consideration. This makes the computations more efficient and reduces the memory load since the treatment of an iso-component requires much more overhead than the treatment of a posting.

We use a standard algorithm from digital imaging to group the same height postings into a connected iso-component. With one scan through the data, left-to-right and bottom-to-top, we label all undecided postings. If a posting has the same height as one of its neighbors that has already been scanned, then the label from the first such neighbor is given to the current posting. Otherwise, a new label is assigned to the posting. Thus postings that belong to different iso-components will clearly have different labels. However, it is possible for postings from the same iso-component to have different labels. Therefore, in parallel with labeling, we create a graph of the equivalence relation between the labels. Two labels are equivalent if and only if their postings belong to the same iso-component. The vertices of this graph are the different labels, and when we encounter a posting that has two same height neighbors with different labels during the scan, we draw an edge between those two labels. All labels belonging to the same connected (in graph theoretic sense) component of this graph represent postings

belonging to one iso-component. We extract the connected components of the graph using a standard breadth-first algorithm, and then we relabel the undecided postings using a single label from the equivalence class for each label in that equivalence class. Thus, at this point we will have the postings in each iso-component marked with a unique label. Next, we extract the boundary of each connected iso-component. By boundary we mean all postings that do not belong but have a neighbor that belongs to the iso-component. The boundary is not necessarily a connected set of postings. Without formalizing our claim mathematically we state that boundary of every iso-component consists of one connected outer boundary and zero, one, or more pieces of connected inner boundaries that are, in essence, boundaries of "holes" in the iso-component. To obtain the boundaries we employ an algorithm very similar to the standard marching squares algorithm (described in the next section), combined with the appropriate flagging of visited postings. Our algorithm has computational cost proportional to the length of the boundary.

Once the boundary is available, we march along each of its pieces, accumulating the number of changes from higher to lower height and vice versa as we go. Summarizing the results from all pieces of boundary, we classify the iso-component as a minimal, maximal, saddle, or regular plateau. In the first three cases, we select a posting on each piece of boundary and store it as a seed for the marching square algorithm together with the local minima and maxima, and saddle points found in the preliminary step described above.

Extract Level Curves

One type of curve that we use to obtain a good approximation to the terrain is the level curve. Level curves are drawn on topographical maps to identify the locations that share a common elevation. The characteristics of individual level curves (for example, shape or curvature) and the relationship between curves that are in the vicinity of each other can provide useful information about the geometry of the terrain. First, let's understand what is a level curve in a continuous setting and then discuss how this application handles level curves in the case of discrete information. Given a bivariate continuous function $F(x, y)$ and letting $S = \{(x, y, z) : z = F(x, y)\}$ be the surface that is defined by the function, let C_λ be the set of level curves at $z = \lambda$ and defined to be the boundary curves of the connected components of the set:

$$Z_\lambda = \{(x, y, z) \in S : F(x, y) > \lambda\}$$

Continuity of $F(x, y)$ implies that the curves of C_λ will be closed, continuous curves. Obviously, there are no points in common between two of these types of sets with different values of λ. Along with this, there is a natural relation between any two level curves with differing λ's: either 1) one encircles the other or 2) neither encircles the other. This inclusion relation will be used later in establishing the Morse structure of the terrain and assigning priorities to the selected level curves. In this application where the data provided is considered to be a grid of uniformly sampled elevation data, the function $F(x, y)$ is assumed to be a piecewise bilinear function that coincides identically with the given data at the grid points. Using this model, function values of $F(x, y)$ not provided in the data set can be interpolated by using four neighboring postings (that form a cell of the grid). By using the bilinear model, the function F and surface S are continuous. The level

curves for this model are known to be polygons or piecewise linear, closed curves and can be represented by storing the vertices and parameterizing the line segments that connect the vertices.

To extract level curves from the discrete dataset, we use a fast, recursive procedure called the Marching Squares algorithm (Lingrad, *et al.,* [9]). Imagining the data as a collection of cells (like a quilt) with the postings at the corners of the cells, the algorithm "marches" from cell to cell determining the path of a level curve as it transverses the grid. Marching Squares works recursively by walking from cell to cell, analyzing the posting values of the four corners of the current cell to determine the direction of the next cell to which to march, and marking the cells for which it has encountered. For a given $z = \lambda$, the algorithm will extract all level curves of C_λ and will terminate when all cells are either known to be above or below the λ threshold or marked during the marching procedure. The speed of this algorithm comes from the fact that searching unnecessary cells is avoided. Using the bilinear model, levels curves are polygonal and the algorithm can identify the vertices (locations where the curve passes from one cell to another) that define the polygon. To see how the Marching Squares algorithm works, consider the following example. Let the values at the corners of the current cell be: NW = 200, NE = 202, SE = 210, and SW = 206. If one is tracing the level curve where $\lambda = 208$ and enters the cell from the south (between the SW (206) and SE (210) postings, $206 \leq \lambda \leq 210$), then the algorithm dictates that the next cell in the march will be to the east since $202 \leq \lambda \leq 210$. Once in the next cell, comparing the four corners of the new cell with $\lambda = 208$ will prescribe the next cell in the march. See Figure 7 for a more complete example (for clarity, the values of the postings are replaced with 0 and 1 for lower and higher,

respectively, in the middle two images). With the assumption that the function defined the surface between postings is bilinear, then the actual coordinates along the eastern boundary can be computed using interpolation.

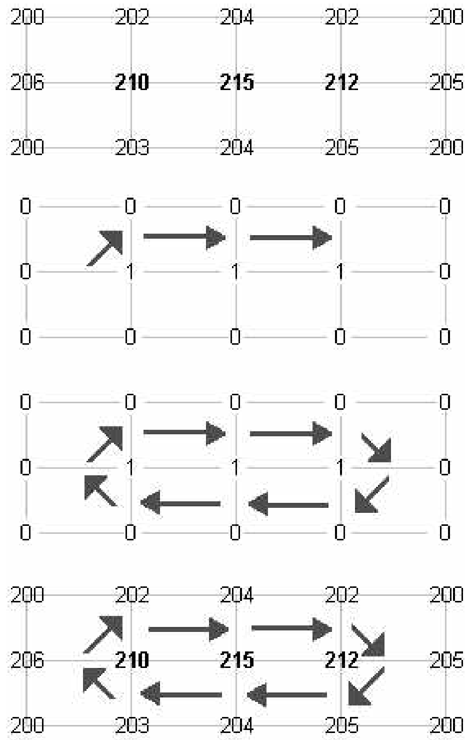
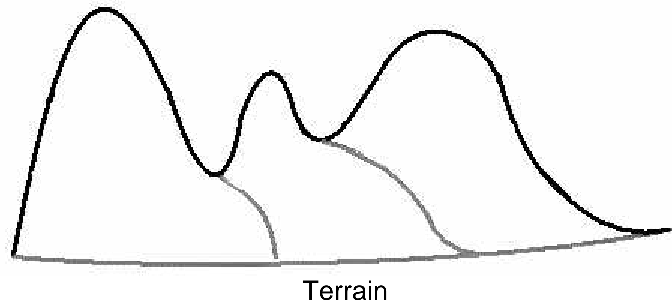
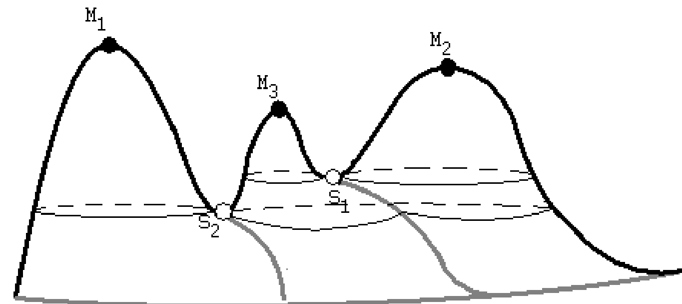


Figure 7: Marching Squares Algorithm

We choose the level curves that pass through the critical points, since those curves form the boundaries of maximal monotone sections in the terrain. The critical points completely identify the level curves by providing both the starting cell and the lambda value for each.

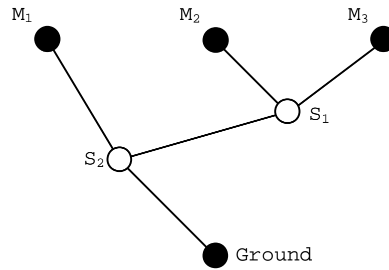


Terrain



Ground Point (global minimum for reference)

Critical Points and Associated Level Curves



Morse Structure

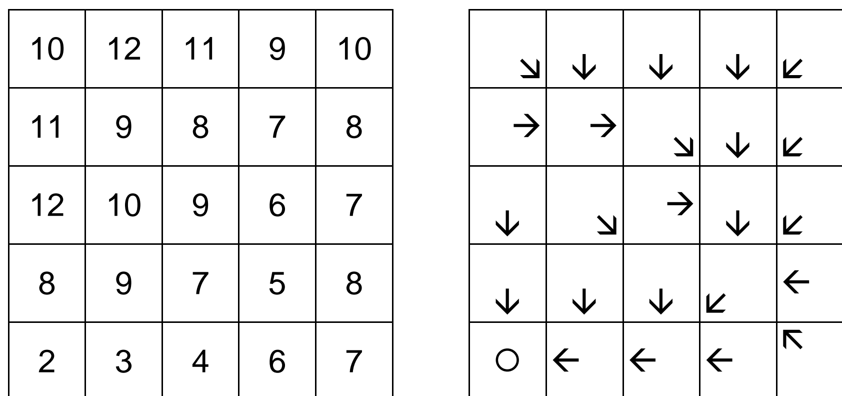
Figure 8: Morse Structure of Level Curves

Extract Ridge Curves and Ravine Curves

To extract ravines, we modified a standard flow accumulation algorithm used in GIS hydrology applications (DeBarry and Carrington, [3]). The same algorithm is used to extract ridges as well by first inverting the dataset.

For each posting, we calculate either global or a local version of flow accumulation. The flow accumulation value is the number of upstream postings whose flow paths pass through the given posting. So we first form a directed graph of the flow paths.

As with finding critical points, forming a flow graph is a straightforward process if adjacent pairs of postings always have different slopes between them. In the graph, each posting is a vertex. The directed edge (*v1*, *v2*) exists if and only if *v1* and *v2* are adjacent postings and the value of *v2* is less than the value of *v1* and the slope between those two postings is steeper than the slope between *v1* and any other neighbor of *v1* lower than *v1*. The edges give an approximation of the path of steepest descent from each vertex (how water would flow in the terrain when poured at any given vertex). See Figure 9. This directed graph turns out to be a tree, with the edges pointing the path to the root. The flow accumulation value of a given vertex *v1* is the number of vertices *v* for which a directed path exists from *v* to *v1*. See Figure 10. The root vertices are the postings at local minima (called sinks in the GIS references) or at the boundary of the dataset.



10	12	11	9	10
11	9	8	7	8
12	10	9	6	7
8	9	7	5	8
2	3	4	6	7

Figure 9: Sample Terrain Data and Corresponding Flow Graph.

0	0	0	0	0
0	3	5	2	0
0	0	0	11	0
1	0	1	15	0
24	21	19	0	0

Figure 10: Sample Flow Accumulation and Resulting Ravine.

To improve that algorithm slightly, we extend it to handle cases where more than one adjacent posting is at the steepest descent from a given posting. In this case, we add an edge to each of the neighbors with steepest descent. Doing so will either join two trees or create a cycle in one tree. Either way, the structure is no longer a forest of trees, but rather a general directed graph. Some care must then be taken when calculating the accumulated flow to avoid considering a vertex multiple times in the accumulation when that vertex has more than one path to a given vertex. In terrain applications, this over counting does not adversely affect the accumulation computation, since it only makes the ridges and ravines more pronounced (artificially inflating the accumulation values which were already above the threshold).

Connected groups of same height postings are handled similarly for this process as they are for extracting critical points. The postings are gathered into iso-components, and the iso-component is then treated as a single vertex in the flow graph. The graph is weighted to reflect this by assigning each vertex a value indicating the number of postings it comprises. A vertex's accumulation is then the sum of all the weights.

The GIS algorithm typically includes additional handling of the sinks, including iso-components with no downslope neighbors (no spill points). The most commonly used

algorithm involves preprocessing the terrain to fill all sinks, raising them until each one becomes part of an iso-component with a downslope neighbor. Thus each posting has a valid downslope path that leads to the boundary of the data. This process can be conceptualized as flooding the sinks to convert them to lakes with an outlet. For ridge extraction, no special handling is applied to avoid sinks.

Schedule Sections for Encoding

After the collection of curves that will be used to render the surface has been identified, the next step in the process is to find the best ordering of the curves for subsequent use within the application. In a simplified description of the overall process, the client will be receiving small subsets of curves per transmission, and, after each new batch of curves is received, the client will generate an approximating surface based on the information that is at hand. Determining a good order in which to send the curves will help in generating a better approximation to the true surface.

The first step in determining the order of the level curves is to construct a tree that is based on the natural relationship between closed level curves. Projecting the set of level curves onto a common plane and considering the interiors of the closed level curves, a relationship between two level curves can be defined based on the interiors being disjoint, equal, or one being a subset of the other.

With the natural inclusion relation that exist between level curves, a tree of level curves is created where a parent node P with a child node C implies that the level curve associated with node P encircles the level curve associated with the node C. The highest nodes of this tree are associated with the level curves that encircle the most number of other level

curves. When the tree includes the level curves that contain the critical points of the surface, the tree contains the same information as the surface's Morse structure.

In contrast, there is no natural relation between the λ values of a parent node and its children. This can be exemplified by the level curves of a volcano where first the lambda values increase as the level curves start at the base and move to the rim, but then decrease once inside the rim. At the same time, the differences in elevation between parent and children level curves are used as a measurement to assist in the ordering process.

Decompose Curves

The curves, both level and ridge curves, are first prepared for multiresolution analysis by splitting each into one or more pieces (*i.e.*, segments) using a sequence of points for the divisions. The splitting is determined by the type of curve, the complexity of the curve, and the curve's intersections with other curves. These points then are the end points of each piece of the curve that results. For an open curve, the two end points must also be chosen and serve as anchor points. The multiresolution analysis is done on each segment of a curve between anchor points.

Points of intersection between this curve and other curves, such as those produced by the drainage structure, are first chosen. Then, for level curves, additional anchor points are selected by adaptive thinning, which we describe in more detail in the next section. If the curve is a closed curve and no points are selected, then an arbitrary point is selected to be the anchor point.

Adaptive Thinning

For each curve a coarse initial approximation is generated using adaptive thinning (Dyn, *et al.*, [8]) to select a small number of points on the curve which, when connected with line segments, constitute a "good" approximate curve. The adaptive thinning algorithm assigns a priority to every point of the polygonal curve. In each iteration, it greedily excludes the point of lowest priority, places it in a priority queue, and reassesses the priorities of its neighbors. In our implementation, we use as priority of a point the Hausdorff error that would be introduced to the current approximation of the curve by the removal of that point. The initial approximation consists of the points in a final segment of the priority queue (*i.e.*, those removed last). The size of the final segment can be a predetermined number of points in the original representation of the curve, or can be chosen as the minimal set of points that achieves a certain error tolerance.

The nature of the adaptive thinning allows us to easily create an initial approximation that is built around a set of points that are *a priori* identified as important (such as the intersections of level curves with ridges and ravines, for instance). Indeed, giving these points the highest possible priority and not allowing their priorities to be changed guarantees that they will be the last to be removed, and therefore their presence will be taken into account when the priorities of the rest of the points are computed.

Multiresolution Analysis

Having established an admissible initial approximation, the server performs a multiresolution analysis of each curve. Namely, it generates a sequence of increasingly more accurate approximations to the curve, each of which is a refinement of the previous

approximation. Recall that the initial approximation is interpolating and so it partitions the curve into arcs whose chords are exactly the line segments forming the approximation. Therefore, we content ourselves with building a multiresolution analysis for every arc independently of the rest of the curve. As coarsest resolution we use the chord of the arc. The second level of resolution is produced by refining the chord and consists of two new chords. And so on, with each level refining each chord by replacing it with two new chords. See Figure 11.

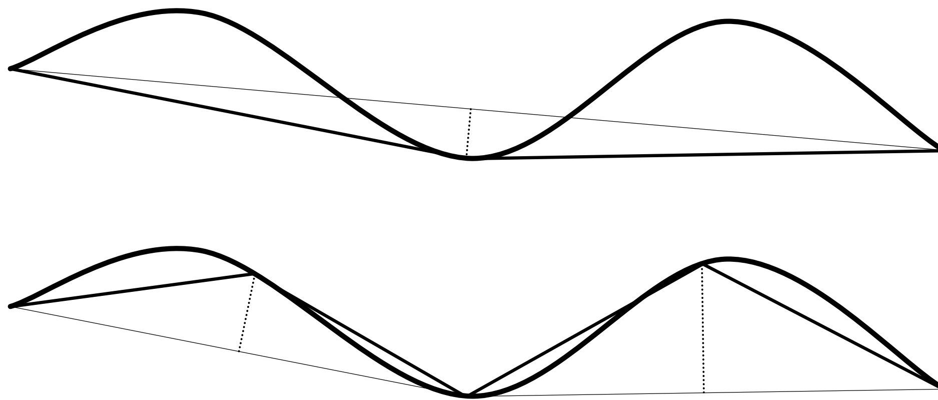


Figure 11: Multiresolution Analysis of a Curve

For level curves, the following scheme is used for refining a chord. The scheme for ravine and ridge curves is similar, but slightly more complex because the curves are not planar. First, we find an intersection point of the original level curve and the perpendicular bisector of the chord. This point is called a refining point of the chord, and the signed distance from the refining point to the chord is called displacement. The refinement of the chord consists of the two line segments connecting its refining point with its two end points. The sign of the displacement indicates whether the displacement is to the left or to the right as one walks the chord in the direction suggested by the

ordering of the endpoints in the data structure. Note that the refinement of a chord is completely determined by the displacement (and the chord itself). To find a refining point we split current arc into two new arcs of almost the same arc length (measured as number of vertices on the arc) and select the one of them that intersects the perpendicular bisector. We iterate the process until an arc of arc length two (*i.e.*, an arc which is an edge of the original curve) is reached. The crossing of this edge and the perpendicular bisector is used as a refining point. It is worth noting that the refining point is not necessarily a vertex on the original curve, but a point on an edge of the original piecewise linear curve. Repeatedly using the above refinement scheme, each arc is represented as a single-root binary tree. The root of the tree is the chord connecting the two end points of the arc, and the descendants are the chords generated by the multiresolution analysis. Each node stores its displacement. The depth (number of levels) of the binary tree is chosen to be the same for all arcs of a level curve but may vary between different level curves. It is determined as the smallest integer exceeding the base 2 logarithm of the maximal number of vertices on an arc where the maximum is taken over all arcs formed by the initial approximation of the curve. This depth is intended to provide resolution of the curve to at least pixel accuracy everywhere. Clearly, if an arc is much shorter than the others it will be resolved to subpixel accuracy, but this will not have detrimental effect on the compression since all displacements on the finer levels will be zeros and thus will never be encoded.

For ravine and ridge curves, the process is the same except that the perpendicular bisector of a chord is a plane and the displacement is therefore recorded as a vector. We use a three-vector (the difference in the three coordinate directions from the midpoint of the

chord to the point of intersection), but it is possible to use a two-vector in the coordinates of the perpendicular plane. Recall that the surface is a function, so the projection of the unit Z vector to the perpendicular plane will always be a non-zero vector and can be used as the first coordinate direction. Using a two-vector will naturally produce better compression than using a three-vector (since, in this case, the number of bits required for each of the individual components will be the same for both methods).

The reconstruction of the finest resolution of the MRA provides an excellent approximation to the original piecewise linear level curve.

3. Communication (Data Encoding and Decoding)

Progressive Transmission

Only the multiresolution analysis of the approximated curves and the basic parameters of the terrain are transmitted. The client converts these into a set of points and curves in a predetermined manner. Interpolation algorithms are used to map these skeletal elements into a terrain (piecewise linear or otherwise). The server is aware of the interpolation method to be used by the client and can adjust the priorities of curves to be transmitted (and thus the ordering of those curves) accordingly.

Transmission proceeds as follows. The basic parameters of the surface are sent first: the dimensions in X, Y, and Z, and the minimum Z value (the minimum X and Y values are taken to be zero). The dimension in Z is not needed exactly; it is sufficient to know how many bits are necessary to represent the largest Z value (in terms of its offset from the minimum Z value). So, in practice, this smaller number is sent instead.

The starting bitplane (the largest j for which $2^j \leq$ the largest magnitude found in the coefficients) must also be sent so that the client knows where to begin.

The Z values of the four corners are sent in full to start, to serve as a base for the terrain. This avoids having noticeable discontinuous artifacts as successive bitplanes are transmitted. We've also tried extending this idea to sending the boundary curves (the intersection of the terrain with the four boundary planes X=0, Y=0, X=*maxX* and Y=*maxY*) as the first four curves (encoded in the same way as the other level curves). We

have left this as an option, possibly to be selected automatically based on the terrain analysis (and therefore we also transmit another bit to indicate whether the boundary curves or just the four corner Z values are sent).

The bulk of the transmission, though, is the multiresolution analysis data for the curves themselves. Each curve is set up by first sending the number of segments it has and the coordinates of the endpoints. For a level curve, the Z value of the curve is sent, to full precision, as another element of the set up for that curve. Other coordinates are sent only to the precision of the current bitplane (or to some predetermined minimum required resolution).

Those other coordinates are updated with each successive bitplane in the same manner as the magnitudes of the multiresolution coefficients. Note that, if the origin is in the southwest corner, this means that the anchor points will move only north and east, if they move at all, as successive bitplanes are sent. At bitplane j, the error for any given coordinate will be at most 2^j.

The multiresolution analysis coefficients are sent for terrain exactly the same way as they are for images, except that the trees are binary trees instead of quad trees. There's one tree for each segment of a curve between anchor points, and they're sent in the same order as the curves are defined (and in the same order as the anchor points are given within each curve). In this case, however, we do not assume that the root exists for every segment. The root for each segment must be explicitly grown, or the segment will simply remain unrefined (*i.e.*, a straight line between its end points).

We send the bits for positions of the coefficients (the tree growth bits) for all trees before moving on to the coefficient magnitude and sign bits. This arrangement is chosen to

isolate the elements that are effectively random. This segregation of the components of the transmission stream helps to maximize the efficiency of any adaptive arithmetic encoder being used.

4. Client (Data Reconstruction)

The client must perform two main tasks. It has to construct the approximated curves from the bitstream data, and it must then render a surface based on those curves.

Reconstruct Approximated Curves

To reconstruct the curves, the client must construct the MRA trees and fix the anchor points for each curve as the curve data is read from the bitstream (updating both the MRA coefficients and the coordinates of the anchor points as more bits are read). To construct the anchor points and the MRA trees, we reverse the process describe above for progressive transmission. After each bitplane is received and processed, the client then stops reading the bitstream and creates a new rendering of the terrain from the portion of data it has received. Then the client reads the next bitplane, makes a new rendering, and so on.

Create Piecewise Linear Triangulation

In this section, we describe one method to render a surface by interpolating from the curve data received. The final surface is piecewise linear, and each linear patch is a triangle. This is efficient both in terms of the processing required to make the triangulation and in terms of the display of the triangulation. Almost all graphics video cards in computers today expect (and are optimized for) rendering such triangulations.

In our system, the client creates a Delaunay triangulation of the points (both the anchor points and the points inserted during the MRA step) projected into the XY plane. A Delaunay triangulation of a set of points (in 2-D) is a triangulation of the set of points with the property that no point in the set falls in the interior of the circumcircle (the circle that passes through all three vertices) of any triangle in the triangulation. For this, we use a program called *Triangle* (Shewchuck [12]), which is a fast and memory-efficient Delaunay triangulation generator. We also preserve the line segments of our reconstructed curves by performing a series of edge swaps on the Delaunay triangulation. This process is also done by *Triangle* when called with the appropriate options.

The resulting terrain is piecewise linear (each triangular patch is flat) and is well suited for rendering on most graphics hardware today via the OpenGL graphics language, the *de facto* standard in graphics programming.

5. Results

In testing the approximation algorithm, as a first step, we have isolated the curve selection and prioritization from the multiresolution analysis. We select a set of curves to start and proceed with just that set of curves. (That is, no new curves are introduced in the middle of the progressive transmission. Including this step that would provide dramatic improvement in the refinement capability at successive levels of transmission, but it is omitted here in order to test and tune the curve selection parameters.).

For selecting ridge and ravine curves, we tested accumulations of 5, 10, 15, 25, and 125 (this is the p_1 parameter). For each p_1 accumulation target, we used a neighborhood of $(p_1)^2$ multiplied by one of 0.125, 0.25, 0.5, 1, and 2 (the neighborhood is the number of cells to look back through when determining accumulation - the number of uphill steps to take on the flow graph. This is the p_2 parameter). For each combination of p_1 and p_2, we used a threshold length of 5, 10, or 15 postings (ridges and ravines who length was less than the threshold were discarded).

For selecting level curves, we used a weighting of 0.01 times the size of the interval containing the height values. We also tried smaller weightings, but those produced systems of level curves that were too tightly spaced. With that system, the approximated curves overlapped too often, and this confused the Delaunay algorithm and produced poor approximations (or sometimes no approximation at all, if the Delaunay algorithm became trapped in an infinite loop).

For each combination of parameters, we selected the curves and transmitted ten bitplanes, computing errors in the L_∞ and Hausdorff metrics after each bitplane. See Figure 12 for a sampling of the combinations of parameters with the best performance (the Hausdorff errors and the L_∞ errors were always very close; the Hausdorff error is naturally never greater than the L_∞ error, and is a better indicator as to the quality of the approximation).

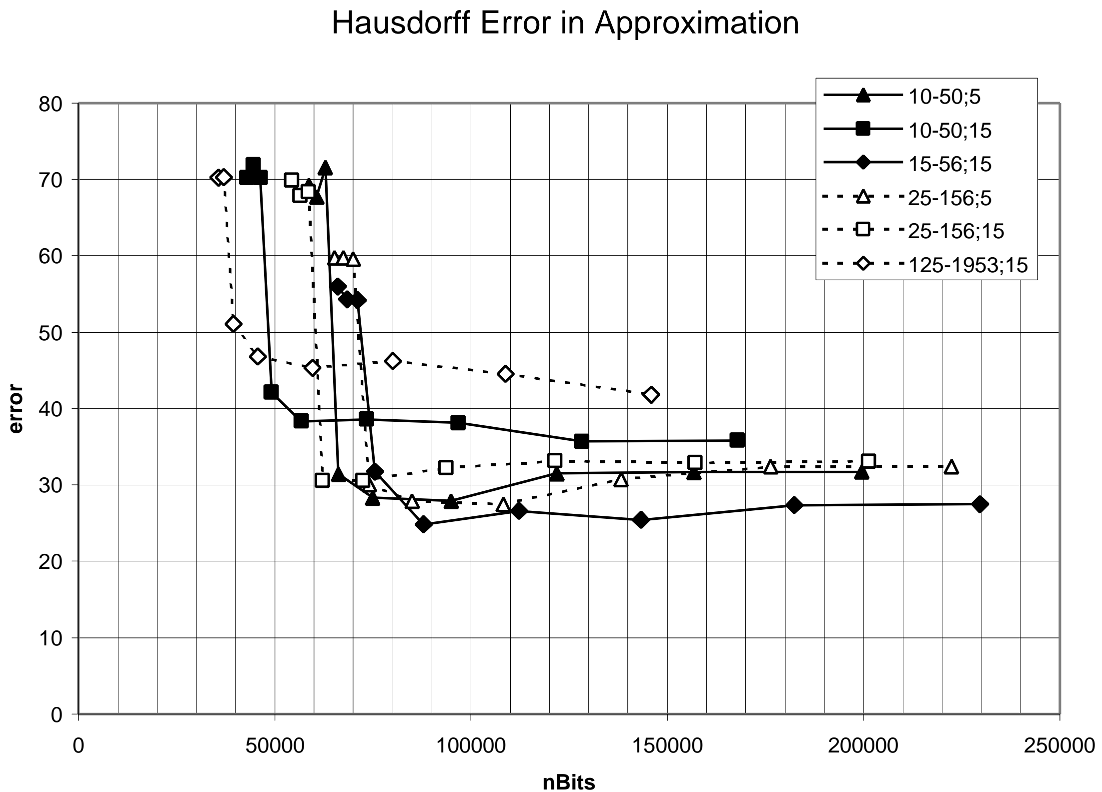


Figure 12: Results

In all cases, the improvement in the approximation leveled out dramatically after bitplane zero, suggesting that bitplane zero should be the last bitplane sent for any given curve and that if additional resolution is needed, then new curves should be sent.

The following figures show images from the transmission marked with ? above, but with fewer level curves (to help distinguish them). First the actual terrain and the actual ridges,

ravines, and level curves selected from it, then the information the client reconstructs after each of the first four bitplanes (bitplane three down to bitplane zero).

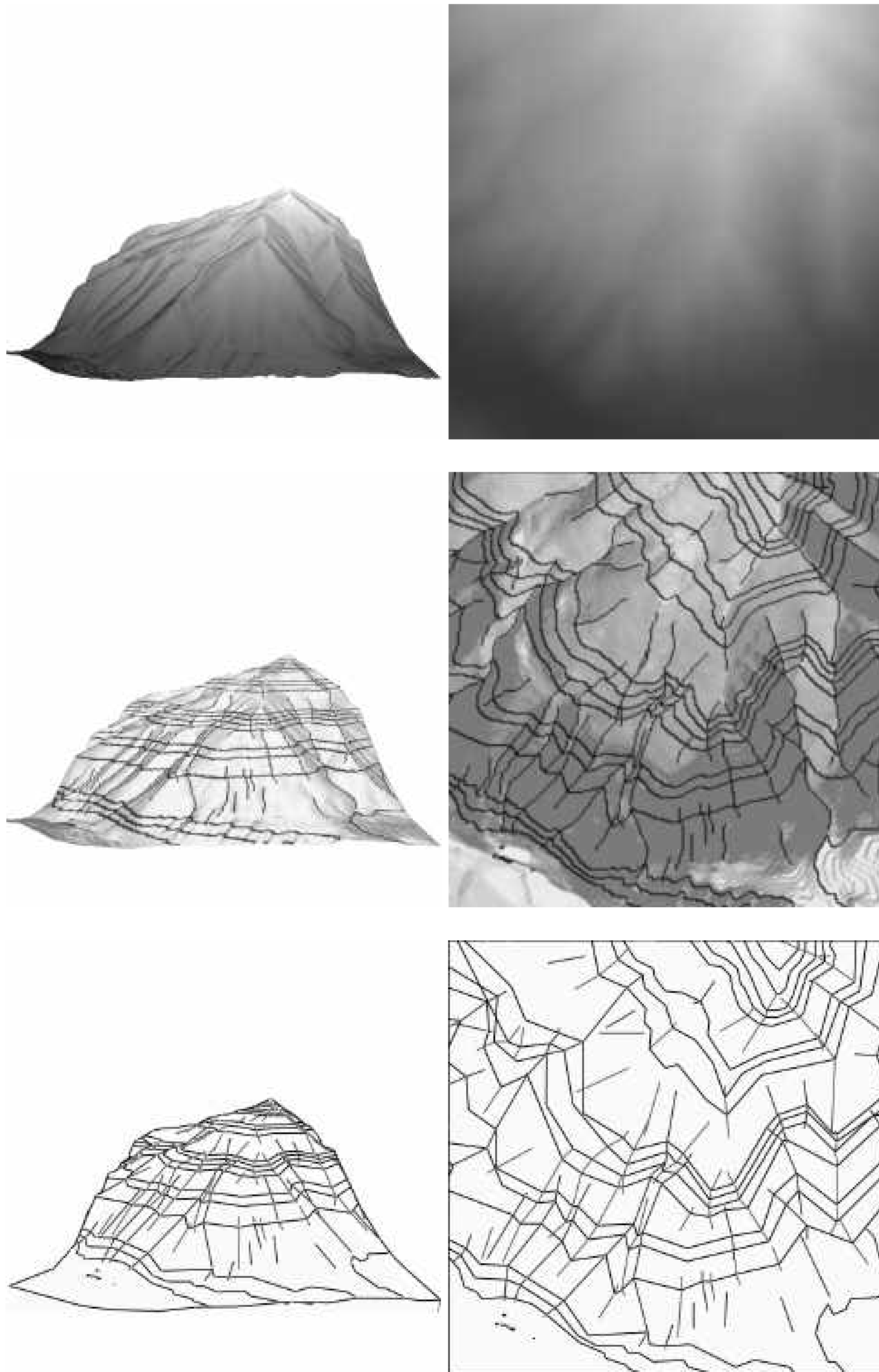


Figure 13: Maple Mountain Dataset, Selected Curves, and Initial Approximation

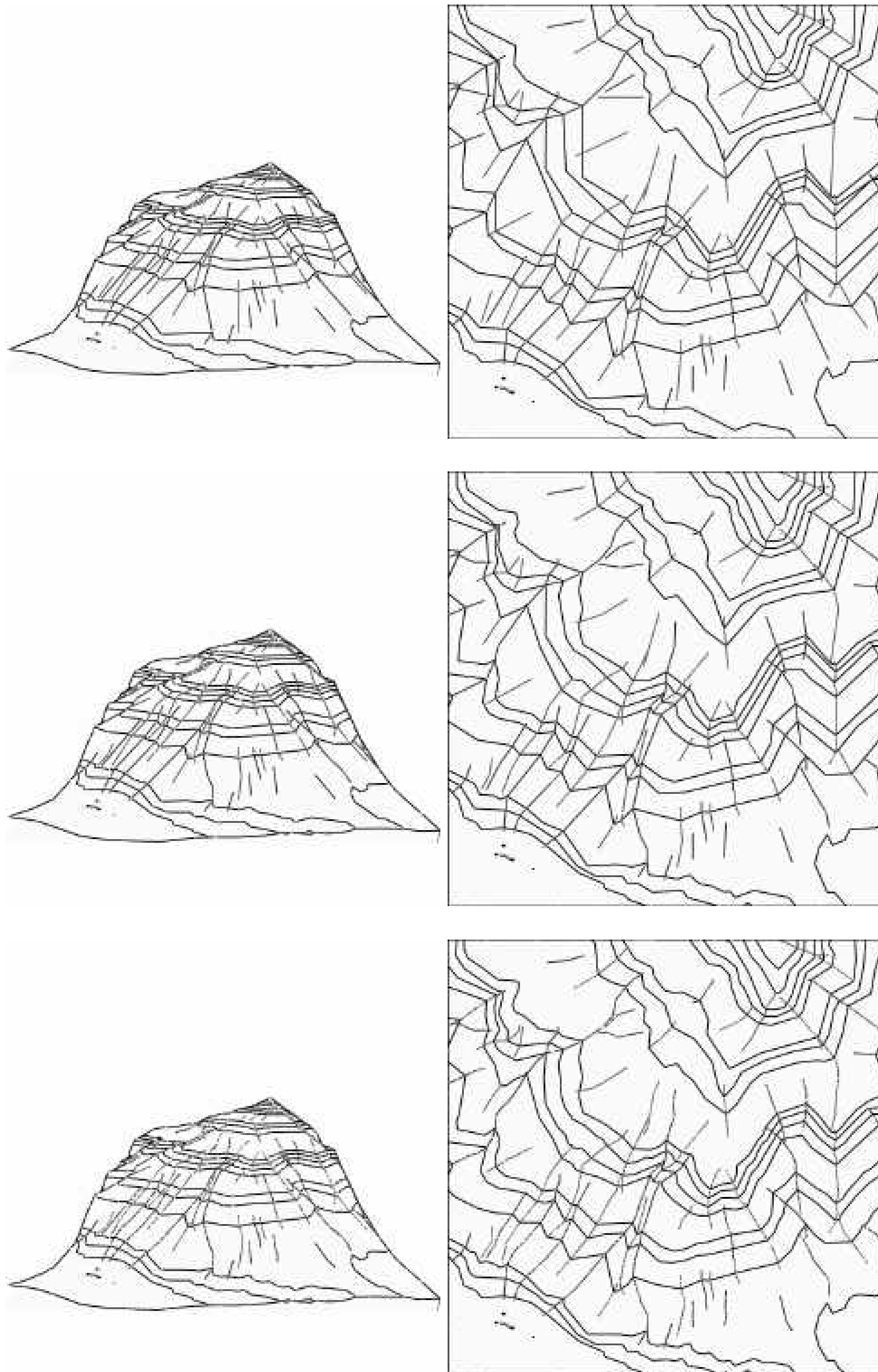


Figure 14: Maple Mountain Reconstruction through Next 3 Bitplanes (2, 1, and 0)

Bibliography

1. Chambolle, A., R. DeVore, N. Lee, and B. Lucier, Nonlinear wavelet image processing: variational problems, compression, and noise removal through wavelet shrinkage, IEEE Transactions on Image Processing **7** (1998): 319-355.
2. Cohen, A., W. Dahmen, I. Daubechies, and R. DeVore, Tree Approximation and Encoding, ACHA **11** (2001): 192-226.
3. DeBarry, P.A., and J.T. Carrington, Computer Watersheds, ASCE Civil Engineering **60** (1990): 68-70.
4. DeVore, R., B. Jawerth, and B. Lucier, Image Compression Through Transform Coding, IEEE Proceeding on Information Theory **38** (1992): 719-746.
5. DeVore, R., B. Jawerth, and V. Popov, Compression of Wavelet Decompositions, American Journal of Mathematics **114** (1992): 737-785.
6. DeVore, R., L.S. Johnson, C. Pan, and R. Sharpley, Optimal entropy encoders for mining multiply resolved data, in *Data Mining II*, N. Ebecken and C.A. Brebbia (eds.), WIT Press, Boston (2000): pp. 73-82.
7. DeVore, R., Nonlinear approximation, Acta Numerica **7** (1998): 51-150.
8. Dyn, N.; M. S. Floater, and A. Iske, Adaptive Thinning for Bivariate Scattered Data, J. Comput. Appl. Math. **145**, no. 2 (2002) 505-517.
9. Lingrand, D., A. Charnoz, R. Gervaise, and K. Richard, The Marching Cubes [online]. Ecole Supérieure en Sciences Informatiques (ESSI): Université de Nice -- Sophia Antipolis. (2002). Available from http://www.essi.fr/~lingrand/MarchingCubes/algo.html; Last accessed on 21 July 2004.
10. Mitchell, J. L. and W. B. Pennebaker, Optimal hardware and software arithmetic coding procedures for the Q-Coder, IBM J. Res. Develop. **32** (1988): 727-736.

11. Pennebaker, W. B., J. L. Mitchell, G. G. Langdon, Jr., and R. B. Arps, An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder, IBM J. Res. Develop. **32** (1988): 717-726.

12. Shewchuk, J. R., Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, *First Workshop on Applied Computational Geometry* (Philadelphia, Pennsylvania), ACM, (May 1996): 124-133. Also available online from http://www-2.cs.cmu.edu/~quake/tripaper/triangle0.html; Last accessed on 1 July 2004.

Appendix: Annotated Bitstream Example

This example was run on the Maple Mountain data set, a 250 × 250 data set with 16 bits per posting. The height values range from 1456 to 2618. The level curves were filtered with a 30-unit tolerance. The ridge and ravine curves were selected using a 156-cell range and a 25-unit accumulation tolerance. Any ridge and ravine curves shorter than 15 postings in length were discarded.

Bits (Sign bits in *italics*)	Interpretation
Dataset Header	
1	Square Dataset
00011111010	Size: 250x250
0000010110110000	Minimum Z: 1456
1011	Num Bits for Z: 11
00001010010	LL Zvalue: min + 82
01100100000	LR Zvalue: min + 800
00010100100	UL Zvalue: min + 164
01000110100	UR Zvalue: min + 564
0 00011	Initial bitplane: 3
0 000	Coordinate Minimum Resolution: 0
1	Send Boundary Curves (BC0 to BC3)
Bitplane 3 Curve Header	
011101	Num level curves: 29 (LC4 to LC32)
100011	Num ridges: 35 (RIDGE33 to RIDGE67)
100010	Num ravines: 34 (RAVINE68 to RAVINE101)
001111	BC0 numInitSeg: 15
01010010	BC0, V0 Z Coor: min + 82
00000000	BC0, V0 Y Coor: 0
01001110	BC0, V1 Z Coor: min + 78
00001101	BC0, V1 Y Coor: 13
.	
.	
.	
10100100	BC0, V15 Z Coor: min + 164
11111001	BC0, V15 Y Coor: 249
0110	BC0 #Level in Tree: 6
0 0000010	BC0 maxBitPlane: 2
001111	BC1 numInitSeg: 15

.
.
.

0000010	BC3 maxBitPlane: 2
01000001111 111111010111	LC4 Z Value: min + 527.98999
000011	LC4 numInitSeg: 3
0	LC4 is a closed curve
10101101 11001011	LC4 V0 Coor: (173,203)
10101101 11001100	LC4 V1 Coor: (173,204)
10101100 11001100	LC4 V2 Coor: (172,204)
0001	LC4 #Level in Tree: 1
1 0000011	LC4 maxBitPlane: –3
Bitfield for 3:	
00000000011 111111010111	LC5 Z Value: min + 3.98999
000011	LC5 numInitSeg: 3

.
.
.

0111	LC28 #Level in Tree: 7
0 0000011	LC28 maxBitPlane: 3
000010	RIDGE29 numInitSeg: 2
11100010 00010010 01101110010	RIDGE29, V0: (226,18,882)
11101010 00011101 01111001011	RIDGE29, V1: (234,29,971)
11101111 00100101 01111111100	RIDGE29, V2: (239,37,1020)
0100	RIDGE29 #Level in Tree: 4
0 0000001	RIDGE29 maxBitPlane: 1

.
.
.

11001000 11111000 00101111100	RAVINE101, V3: (200,248,380)
0101	RAVINE101 #Level in Tree: 5
0 0000000	RAVINE101 maxBitPlane: 0

COEFFICIENT TREE GROWTH BITS

000000000000000	BC0's 15 segments are not further refined
000000000000000	No growth on BC1's trees, either
000000000000000	Nor on BC2
000000000000000	Nor on BC3
000	Nor on LC4

.
.
.

1010000000000000000	LC16's first segment has a two-node tree
000	Nothing on LC17
0100000000000100000000	LC18's 2nd and 12th trees have roots
00	Nothing on LC19

.
.
.

COEFFICIENT UPDATE BITS

0	LC16's 1st seg. root node is zero. The right child is 8.
	LC18's 2nd and 12th seg. root nodes are 8.

.
.

.

10000000011 Sign bits for all nodes (even ones with zeros)

LC16's 1st seg. root node is negative (but < 8, so 0 for now)

LC16's 1st seg. right child is positive 8

The two roots in LC18 are positive 8

and so on

.
.
.

At the end of bitplane 3, we have transmitted 18719 bits (that works out to 0.3 bits per posting, or 1.87% of the original data). The client's reconstruction has a Hausdorff error of 79.52.

Bitplane 2 TREE GROWTH BITS

00010000000100000010100 LC0's 4th, 10th, and 15th segs have roots, and the 15th also has a right child

.
.
.

11000100000000000000000 LC16's existing 1st seg. tree is expanded to

the rest of the segs are not refined

.
.
.

At the end of bitplane 0, we will have transmitted 29077 bits (that works out to 0.47 bits per posting, or 2.91% of the original data). The client's reconstruction will then have a Hausdorff error of 33.92.