An Algorithm for Continuous Resolution Polygonalizations of a Discrete Surface

David C. Taylor and William A. Barrett
Detparment of Computer Science
Brigham Young University
Provo, Utah, 84602

e-mail:dct@newt.cs.byu.edu
Telephone: 801-378-7430

Abstract

An algorithm for polygonalizing a discrete surface (height grid) is presented. A quad tree is used to create a bottom-up, irregular rectangular polygonalization of an input height grid by merging homogenous regions starting at the pixel level and working up the tree. A homogenous region is composed of any number of smaller regions that differ within a user specified error tolerance in gradient and that fit together to form a rectangular polygon. As rectangular polygons are merged, a polygon tree (polytree) is built which describes the merge process and can be used to rapidly locate neighboring polygons. The polytree is used to describe the discrete surface at multiple levels of resolution. After all possible regions have been merged, the rectangular polygons are triangulated to eliminate gaps created by the irregular intersections of the rectangles. This algorithm automatically preserves critical lines, even with coarse polygonal representations. Continuous resolution can be achieved through the use of TIN morphing between the discrete levels of resolution computed by the algorithm. Additionally, parallel simulations indicate that this algorithm can achieve maximum speedups of $O(\sqrt{n})$ to $O(n/\log n)$, where n is the number of nodes in the bottom level of the tree.

Keywords: TIN, terrain, morph, hierarchical, parallel

1. Introduction

The purpose of polygonalizing a discrete surface is to reduce the complexity of its representation, thereby allowing rapid access to parts of the surface (in a graphical editing system) and rapid rendering of the surface (in real-time environment simulations). One of the major drawbacks of polygonal representation is the loss of realism through the use of coarse representations.

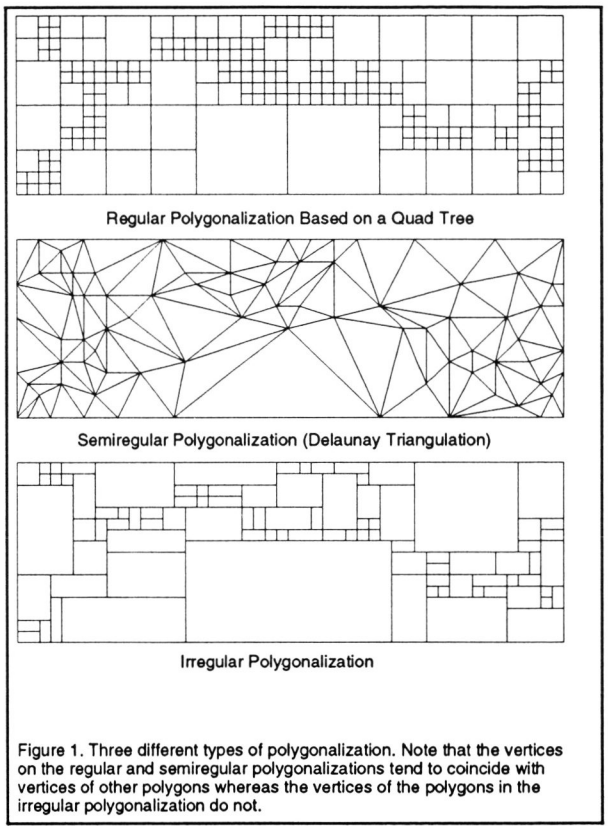


Figure 1. Three different types of polygonalization. Note that the vertices on the regular and semiregular polygonalizations tend to coincide with vertices of other polygons whereas the vertices of the polygons in the irregular polygonalization do not.

If the set of polygon vertices is chosen correctly, the polygonal representation will represent the object much more realistically with a given number of polygons than if the set of polygon vertices is chosen at random. Any algorithm which attempts to optimize the set of polygon vertices must pay particular attention to the preservation of critical lines. Critical lines are parts of the object with sharp discontinuities in gradient, such as ridges and peaks in mountainous terrain. The human visual system relies heavily on critical lines for object recognition and

depth perception.
Past work on polygonalization has concentrated on triangulated irregular networks (TIN) and regular hierarchical polygonalization. TINs are usually based on the Delaunay triangulation [TARVYDAS 84]. The advantages of the Delaunay triangulation include its local adaptability to varying point densities and the fact that the same triangulation is produced for a given set of points regardless of the starting point. However, this type of triangulation is insensitive to critical lines unless points are manually selected on critical lines before triangulation begins [MCCULLAGH 82], [FOWLER 79]. Changing the number of polygons in an object (i.e., for zooming in on terrain) requires a complete re-triangulation of the object or a nonregular hierarchical structure [SCARLATOS 90]. Also, balancing loads for multiple processors requires excessive communications overhead in this approach.

Regular hierarchical polygonalizations are intrinsically parallelizeable because of local information flow patterns, but the top-down approach generally used produces many more polygons than a TIN for a given level of approximation to the terrain [SAMET 88], [BERT 81], [BARONTI 90]. Figure 1 shows examples of a regular hierarchical polygonalization, a TIN, and an irregular rectangular polygonalization.

The polygonalization algorithm described in this paper preserves critical lines and is highly parallelizeable. The algorithm relies on a bottom-up approach to merge similar adjacent polygons. Critical lines are preserved by merging polygons only when they have similar gradients. This process begins at the most local level and gradually extends to encompass the entire height grid. This local to global approach results in massive parallelism in the merge process, since the majority of polygon merges are done at the local level without concern for surrounding data. The user can specify how closely the polygons must match the actual data using a number called the error tolerance (allowable difference between polygons to be merged). By interpolating between polygonal grids of two different error tolerances, continuous resolution can be achieved. The three salient features of this algorithm are completely automated polygonalization to continuous levels of resolution, critical line preservation, and massive parallelism. Each of these are discussed in this paper.

The parallel merge algorithm and the TIN morphing technique are discussed in Section 2. Results of this algorithm are presented in Section 3. Section 4 summarizes current research efforts and future work.

2. Hierarchical Polygonalization of a Discrete Surface

2.1. Data Structures

The polygonalization algorithm requires a height grid as input. A height grid is a two-dimensional array where each entry represents an altitude at a particular point. Height grids can be obtained from various sources ([PETERSEN 90], [JEFFERY 87]). The height grids used in this work are represented by two-dimensional arrays. Natural terrains from USGS DEMs and topographical maps are used.

A standard quad tree data structure is used as a processing architecture for polygon merging (Figure 2,

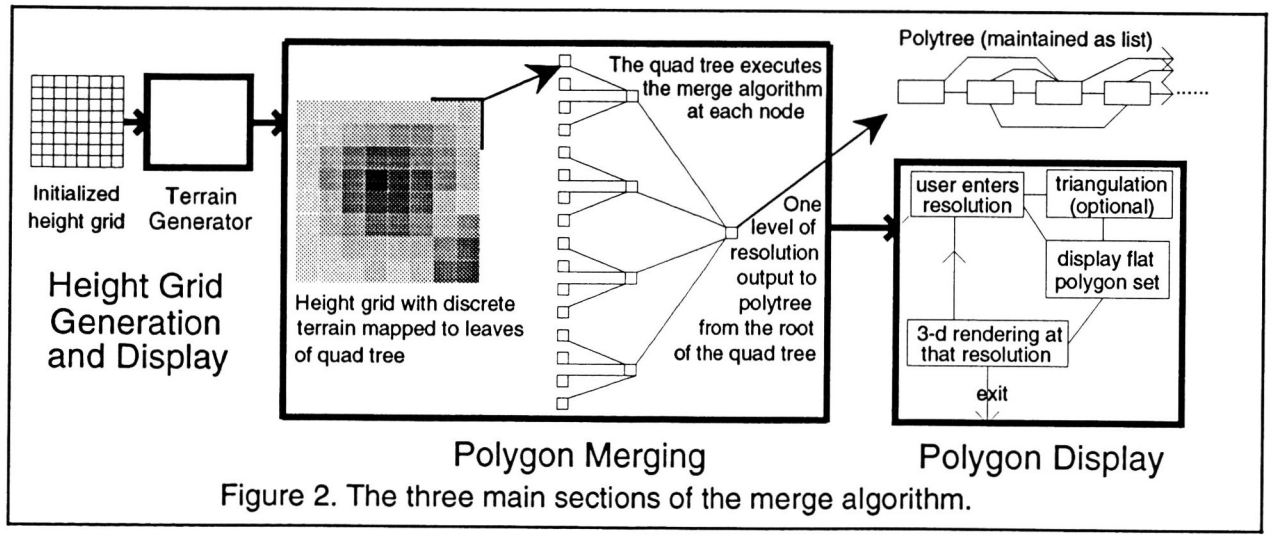


Figure 2. The three main sections of the merge algorithm.

middle). As shown in Figure 2, the quad tree initially has leaves attached directly to the polygons formed by each pixel of the height grid. Each of the nodes in the quad tree maintains a list of polygons in its area of influence. The area of influence of a node is defined as that part of the height grid to which the node's descendant leaves are attached. At the beginning of the merge phase, the polygon lists of all interior nodes are empty. The merge algorithm described in Section 2.4 is applied to each node whose sons all have non-empty polygon lists. This process is repeated until the root node is reached. The result is that each node of the quad tree has a list of polygons for its area of influence.

The quad tree structure is especially useful in operations such as finding neighboring polygons. As the merge algorithm is applied at each node, a structure called a *polytree* (Figure 3) is constructed. Each node in the quad tree has its own polygon list which forms part of the polytree. As shown in Figure 3, the father pointers in the polytree point to polygons in the lists of quad tree nodes at higher levels of the quad tree. These polygons are the result of merging one or more polygons from lower levels of the quad tree. For example, the father pointers of polygons 7, 8, 12, and 13 point to polygon 3 which was created by merging 7, 8, 12, and 13.

Finding neighboring polygons is an important part of the merge process on every level. Figure 3 also illustrates this process. To find the right neighbor of polygon 7, the algorithm is given a point immediately to the right of the right edge of polygon 7. The quad tree is descended until a quad tree leaf node or an interior node that contains this point (which is inside the right neighbor of 7) and has a single polygon in its list is located. The single polygon in the list of this node is a descendant of polygon 8 by virtue of its location on the height grid. Polygon 8 is located by following the father pointers up the partially constructed polytree to a polygon in the list of the current quad tree node. The advantage of this kind of search is that it requires only O(log n) time, where n is the number of nodes in the quad tree. It is possible to search the polygon list of the quad tree node to find X without using the polytree, but this is sequential and does not take advantage of available gains. Non-hierarchical networks must rely on more complex pointer schemes or sequential searches to find neighbors.

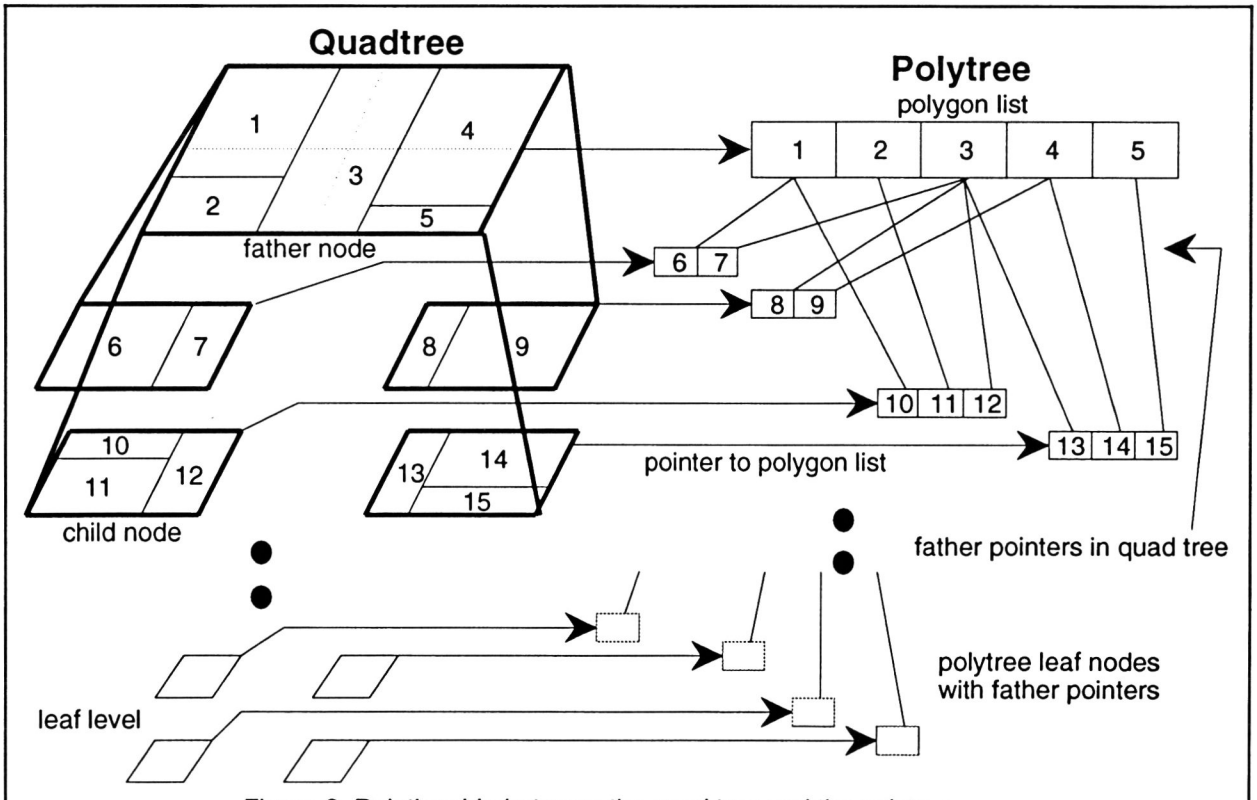


Figure 3. Relationship between the quad tree and the polytree.

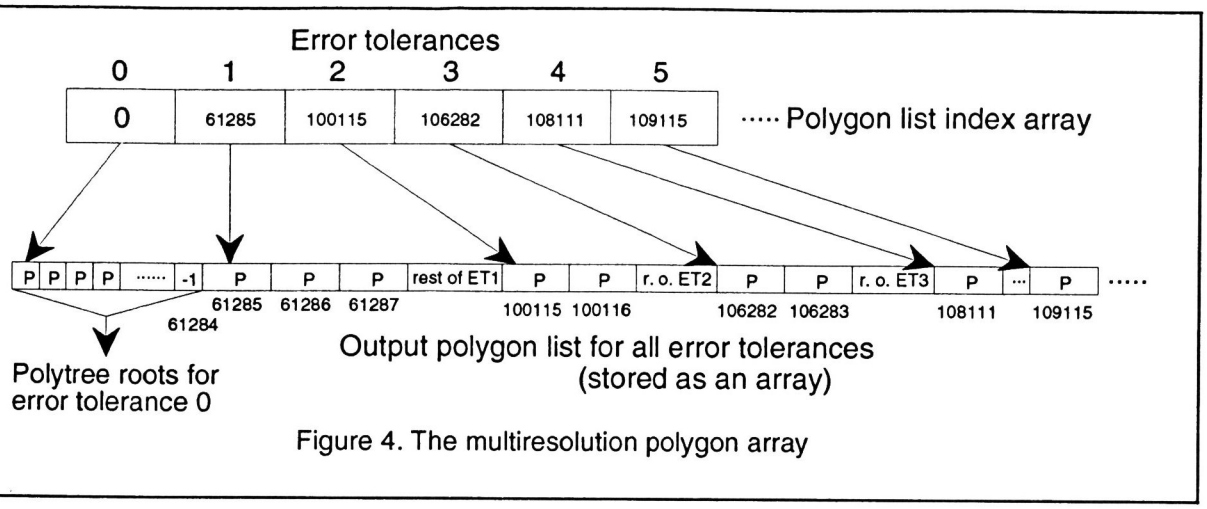


Figure 4. The multiresolution polygon array

When the merge phase is finished, there are two trees in memory: the quad tree and the polytree (Figure 3). The quad tree leaves point to the leaves of the polytree, and the polytree describes the merge process. Following the father pointers of a leaf polygon up through the polytree will bring one to a polygon in the list of the root node in the quad tree, which contains a list of all the "roots" of the polytree. This list is the final polygonal description of the height grid at a given error tolerance.

The output multiresolution polygon array is shown in Figure 4. This output list consists of two arrays. The first array is an index into the second, where all the polygons are stored. For each computed error tolerance, an integer stored in the first array specifies the starting position of the corresponding list in the second array. The second array is an array of polygon structures. This array is filled with the polygon list in the root of the quad tree after each merge phase is completed for a specified error tolerance.

2.2 Homogeneity Criteria

The homogeneity criteria or measure of similarity used to merge neighboring polygons into a single, larger polygon is a function of the gradient. The X gradient is the average difference between the altitudes at the X limits of the polygon divided by the X dimension of the polygon (*xdist*) and the Y gradient is computed similarly. The difference between two polygons is simply the sum of the difference between their X gradients and their Y gradients. Any pair of adjacent polygons whose difference is less than the error tolerance specified by the user is considered mergeable, as long as the resulting polygon can still be described by only four data points as shown in Figure 5. This means that either the X limits or the Y limits of the two polygons must be equal.

The raw gradient measure is modified in such a way that critical lines are preserved. Instead of dividing by the X and Y dimensions of the polygon to obtain a slope as a measure of similarity, the absolute altitude change in the X and Y directions is used, as shown in the following equations which correspond to Figure 5.

$$Xgr = ((z2-z1) + (z3-z4)) \quad (1)$$

$$Ygr = ((z4-z1) + (z3-z2)) \quad (2)$$

$$\text{Grad Diff} = |Xgr_{P1} - Xgr_{P2}| + |Ygr_{P1} - Ygr_{P2}| \quad (3)$$

In Figure 5, polygon 1 has an X gradient of 7 and a Y gradient of -9, and polygon 2 has an Xgr of 15 and a Ygr of -11. The gradient difference is 10, resulting in a merge if the error tolerance is greater than 10. This modification has little effect on the accuracy of the measure because adjacent polygons that satisfy the "X or Y limits equal"

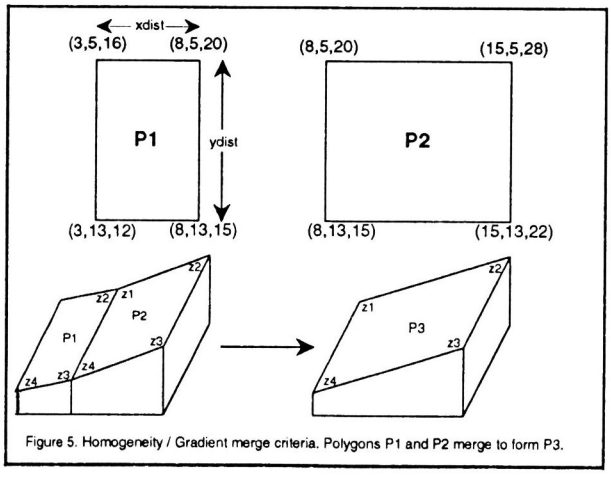


Figure 5. Homogeneity / Gradient merge criteria. Polygons P1 and P2 merge to form P3.

rule are usually either very close to or exactly the same size. The most important effect of this change is to preserve major features at high levels of the tree. This happens because *not* normalizing the absolute difference to a slope or surface normal allows large polygons whose surface normal directions differ by a certain amount to be much less similar than two smaller polygons whose surface normal directions differ by the same amount. This results in a greater resistance to merging already large polygons together, especially at low error tolerances. This ensures that critical features, such as ridgelines and canyons, do not get erased simply because the land surfaces on either side are uniform. Also, this procedure prevents the formation of large, highly nonplanar polygons. Of course, with sufficiently large error tolerances even this extra protection disappears. Even with very high tolerances, this algorithm still shows a remarkable tendency to preserve major topographic features.

2.3 The Merge Process

The merge phase generates a hierarchical description of the object surface. This hierarchical structure contains sufficient information to describe the object surface at varying levels of detail. Each node in the quad tree used to generate this structure consists of a bounding polygon and a list of contained polygons resulting from merges on lower levels, where lower levels correspond to higher levels of detail. The merge phase consists of applying the merge algorithm to each node of the quad tree whose sons have been merged until the root node is reached. The merge procedure for each node is described in part 2 of the algorithm below.

At the beginning of the merge phase, the height grid is mapped to the leaves of the quad tree. The height grid is a surface discretely represented at 65536 points. Each quad tree leaf node is assigned four of these points, which are represented as four unit (1x1) polygons of uniform altitude. It should be noted that larger grids such as 1201x1201 DEM files are handled by subdividing the grid into 256x256 chunks and polygonalizing each chunk separately, then combining the roots of each of the polytrees as if each root were a child node of a larger tree. Likewise, irregular grids are handled by embedding them in a 256x256 grid with points outside the irregular grid set to zero altitude.

The goal of the merge phase is to create a tree of polygons (polytree) that allows leaf polygons to be merged into as few polygons as possible for a specified level of error. The merge algorithm is applied to the height grid for each error tolerance of interest (a). In the merge algorithm, part 2 is initially called with the root node (b). Recursive calls are made to *merge* until a node is reached whose sons all have nonempty polygon lists (c). At the beginning of the merge phase, only the leaf nodes have nonempty lists. The root node is the last node to be merged. When merging a node, the algorithm first gets access to the polygon lists of the sons (d). The algorithm then checks the borders of the sons' areas of influence for mergeable polygons (e). The algorithm proceeds in a specific order when merging polygons from the four sons of a node. As shown in Figure 6, the first step is to merge polygons common to border A between son 1 and son 2. Next, the polygons adjacent to border B between son 4 and son 3 are merged. Finally, the polygons common to border C between sons 1 and 2 and sons 4 and 3 are merged (12-43 merge). This merge algorithm relies on the fact that the only possible merges exist along the boundaries between child nodes (Figure 6), because all possible interior merges are completed at lower levels in the tree.

The two steps to merging are (1), identify polygons

Multiresolution Polygonalization Algorithm	
1. Construct the height grid and initialize structures	
read_terrain(height_grid)	Read in a discrete terrain from a file
init_quadtree(root)	Attach the quad tree to the height grid and set all polylists in each node to NULL
2. Merge the polygons to the root of the quad tree at each error tolerance	
a. for (each error tolerance level) [a-j]	Call the merge control procedure with a pointer to the root of the quad tree
curnode <-- root	
b. merge(curnode)	
merge(curnode): [c-h]	
c. If curnode->polylist is not NULL	This node has already been merged
return	
for (each son of curnode)	Make sure sons are merged before merging curnode (recursive)
merge(curnode->son)	
d. get_plists(curnode->sons)	Obtain the merged sons' polygon lists
e. merge_plist(curnode)	Merge this node (as follows)
(repeat for each border A,B, and C)	This procedure is similar for each border.
P <-- first polygon along border	Find first polygons along border
Q <-- matching polygon across border	
(repeat for all polygons on each border)	
f. If bounds_check(P,Q) and similarity_check(P,Q)	If a pair of polygons passes the two tests then merge them, else record the fact that the merge was unsuccessful.
Make_new_polygon(P,Q)	
else	
g. set_neighbor_flags(P,Q)	
h. P,Q <-- Next polygons down border	Check all contiguous combinations
i. Copy_polygon_list(root)	Copy merged list in root of quadtree for this error tolerance into the multiple resolution polygon list
j. for (each node in the quad tree)	
If (there is >1 polygon in the node's list)	Prepare the quad tree for merging at the next higher level of error tolerance
Delete_Polygon_List(node)	
3. Display the polygonalization	
get_user_error()	Find out the level of error requested by the user
draw_flat_polygon_set()	Show bounds of the polygons by drawing them as rectangles from a top view
render_polygon_set()	Send the appropriate polygon list to the display engine for 3-D rendering

along the boundaries of child nodes, and (2), perform the two merge tests on each adjacent pair of boundary polygons. Adjoining polygons are merged if they satisfy both the bounds check (their combination produces a rectangle) and the homogeneity criteria (f). If an attempt to merge two polygons fails, a flag is set in the polygons to remember this fact for future processing (g). All possible combinations of adjacent polygons along the borders are tested for mergeability (h). Once all the boundaries have been checked for possible merges, the current father (node being merged) goes into "child" mode so that its father may begin the merge phase.

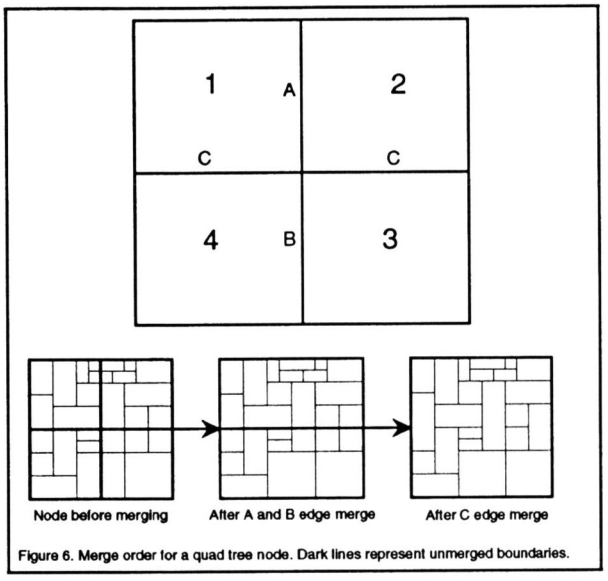


Figure 6. Merge order for a quad tree node. Dark lines represent unmerged boundaries.

A single pass of the merge algorithm produces a set of polygons that describe the surface at a single level of detail (i). This algorithm is used to polygonalize at multiple levels of resolution. The approach taken is to apply the basic merge algorithm for each error tolerance, and not re-merge nodes which have only one polygon in their lists after the previous merge pass (j). This procedure is surprisingly efficient, given that on the average over 80 distinct levels of error tolerance are computed. The algorithm derives its efficiency from the fact that if a quad tree node contains a single polygon after the merge phase on a given level of error, any merges using higher error tolerances will also result in a single polygon remaining in the node. This fact makes it possible to avoid repeating work already done at lower levels of error tolerance and increases the efficiency of the algorithm considerably. (It takes about 2.5 times as long to generate the entire multiresolution polytree as it does to generate the zero tolerance structure in the single resolution case). This process continues until there is only one polygon in the root after a merge. At that point, any higher error tolerances would simply generate that same polygon, so further merges are pointless.

2.4 Extension to triangular polygonalization

When the irregular rectangular polygons are used to render the terrain, gaps appear at points where two rectangles are adjacent along the side of a third. The solution is to triangulate the set of points formed by the vertices of the rectangles to form a TIN [BARR87]. Triangulation is simplified considerably by using a data set such as this, because no triangle ever crosses a boundary of a rectangle. As a result, each rectangle can be independently triangulated. Vertices that lie along the boundary of each rectangle are located in a local fashion using the polytree and the neighbor finding procedure describedin Section 2.1. This means that this triangulation is much more parallelizeable than triangulation on a standard TIN. The Delaunay triangulation is applied to the collection of points found for each rectangle.

Through experimentation, we found that an irregular rectangular polygonalization and a triangular polygonalization of a given terrain represent that terrain to the same level of accuracy with about the same number of polygons. This is advantageous, as triangles are guaranteed to be planar and occupy less memory per polygon. When an irregular rectangular polygon set is triangulated, about 3 times as many triangles are generated as there were rectangles. Since the accuracy of a triangular set is the same as a rectangular set of the same size, this means that a triangular polygonalization at a given error tolerance can be completely represented by a rectangular polygonalization that contains about one third as many polygons. This is equivalent to a 2.5:1 lossless compression of the image, as determined by tests on numerous synthetic and actual terrains.

Most importantly, one of the main problems facing conventional algorithms that rely on the Delaunay triangulation is the selection of data points that preserve important features. The rectangular polygon set preserves critical lines *automatically* by using smaller polygons in areas of high curvature. The irregularity of the rectangular set allows rich variation in the orientation of critical lines, since every 2-edge intersection point in the rectangular set provides a vertex for triangulation. Therefore, the set of vertices defined by the rectangles

provides the type of data set needed by a triangulation algorithm, and in a way that allows highly parallel triangulation (normally a sequential operation). When used in this way, the polygonalization algorithm of Section 2 becomes a massively parallel feature detector for finding critical lines in an image represented by a height grid. This in itself is an important contribution of this work.

Figures 7-9 demonstrate how critical lines are preserved even at high error tolerances. While Figure 8 lacks the surface detail found in Figure 7, major features (such as ridges and canyons) are preserved.

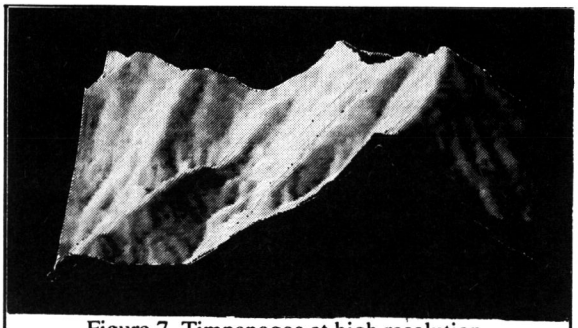


Figure 7. Timpanogos at high resolution.

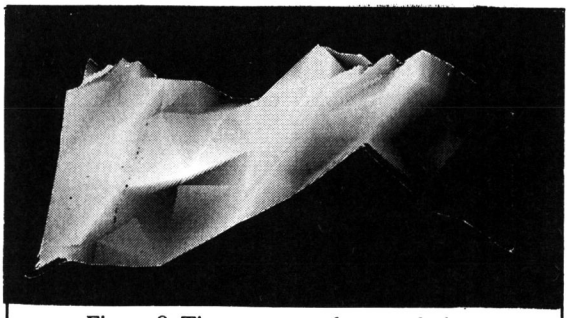


Figure 8. Timpanogos at low resolution.

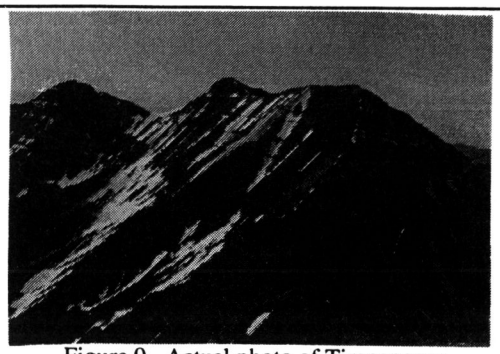


Figure 9. Actual photo of Timpanogos

2.5 Continuous resolution from TIN morphing

The merge algorithm produces a polygonalization for a given level of error tolerance. When two polygonalizations done at different error tolerances are rendered, it is easy to see visual differences caused by noncoincident polygons. This effect is unavoidable unless the error tolerances are close enough to make the difference negligible. This solution is not practical, as thousands of polygonalizations would have to be stored for a single terrain. This problem can be solved by interpolating between two meshes of differing error tolerance. This process is called TIN morphing.

After the polylists for each level of error tolerance are built in the merge phase, they are linked together from low error tolerance to high error tolerance with pointers from polygons in polylists of lower error tolerance to positionally coincident polygons in polylists of higher error tolerances. For example, a polygon with XY coordinate (0,0) that is in the polylist of error tolerance T will point to the polygon with XY coordinate (0,0) that is in the polylist of error tolerance T+1, as long as no polylists with error tolerances between T and T+1 exist. If one polygon in list T coincides with more than one polygon in list T+1, the polygon in list T is split and each of the resulting polygons are inserted into the list of T. Very few additional polygons are created by this operation.

Once pointers exist between levels T and T+1, a

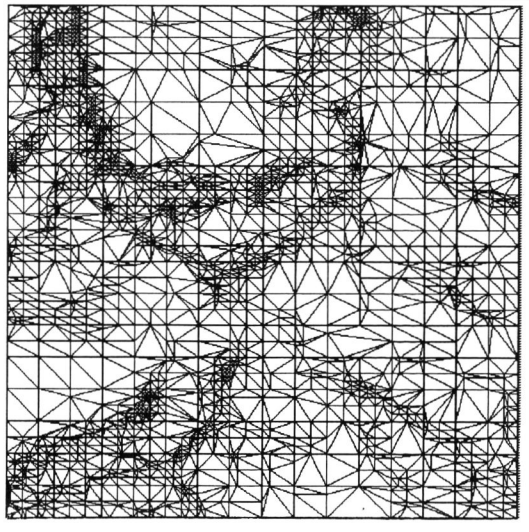


Figure 10. Interpolated triangular mesh.

Continuous Resolution Rendering Algorithm

1. Construct multiresolution polytree (Multiresolution polygonalization algorithm)

2. Get the requested error tolerance from the user (in feet) = utol

Set A is the first generated polygonal set with tolerance atol <= utol.
Set B is the first generated polygonal set with tolerance btol >= utol.

The vertices in set A and set B will be interpolated to find a set U with as many vertices as set A and polygons similar to both sets.

3. Generate a polygon set U between sets A and B

```
i = (utol-atol)/(btol-atol)
for each polygon a in set A
    find the enclosing polygon b in set B
    t = the list of vertices on the top edge of a
        (includes corners)
    l = the list of vertices on the left edge of a
        (does not include corners)
    b = the list of vertices on the bottom edge of a
        (includes corners)
    r = the list of vertices on the right edge of a
        (does not include corners)
    do ft,fl,fb,fr similarly for the father
    (polygon b)
    for each vertex in t
        find the closest vertex in ft
        insert a vertex in ut with coordinates (X,Y):
            X = tX + (ftX-tX)*i
            Y = tY + (ftY-tY)*i
    perform the same operation for b,r, and l
    Starting with the three upper leftmost vertices,
    triangulate the set {ut,ul,ur,ub}.
```

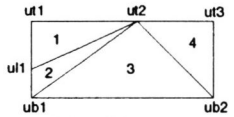
A linear interpolation is used

Use the father pointers previously computed

Lists l and r may be empty.

The lists ut,ul,ur, and ub are now created.

The Z coordinate can be interpolated or retrieved from the height grid at position (X,Y)



Order of Triangulation

4. Render the list of triangles

mapping from the vertices of T to the vertices of T+1 can be created. This process is also parallel, since no vertex from level T that is on the border of or inside a polygon of level T+1 will map to a vertex outside that polygon. This considerably reduces the number of possible mappings, resulting in a much faster algorithm. The X and Y positions of each vertex are linearly interpolated from the low error tolerance mesh to the high error tolerance mesh, and the Z position is taken from the original height grid. The Delaunay triangulation is applied to the collection of points found and interpolated for each rectangle on level T.

This morphing technique provides visually continuous resolution, making a limited number of polygonalizations do the work of an infinite number of polygonalizations.

3. Results

3.1 Data Acquisition

The two datasets used in this paper were acquired from topographical maps. The Timpanogos data came from USGS 15-minute series maps with 40-foot contour intervals, and the Banff data came from the equivalent Canadian series with 100-foot contours. In both cases, digitized maps were converted to discrete height grids using a morphological contour interpolation program [BARRETT 94 (in review)]. These height grids were used as input to the polygonalization program to produce the results shown here. The Timpanogos grid is a single 256x256 area (1km x 1 km) along the Timpanogos ridge. The Banff data is a 1700x2787 area (17 km x 12 km) centered on the northern end of Mt. Rundle. The data points in the Banff data are spaced about twice as far apart as those in the Timpanogos set, resulting in an increased area of coverage and resulting loss of detail. Sixty polytrees are merged to form the complete Banff mosaic, containing about 4.2 million polygons at the lowest level of tolerance.

3.2 Parallel simulation

One of the advantages of the polygonalization algorithm compared to other approaches is that it is massively parallel. This is because the polygonalization procedure has been reduced to a local problem by removing global geometric constraints on the location of the vertices. The quad tree and polytree structures are used to exploit this massive parallelism.

In the simulations of parallelism, it is assumed that each node of the quad tree is replaced by a processor and the links of the tree are replaced by dedicated communication paths from processor to processor. There are no intralevel data dependencies, so no processor requires data from another processor on the same level to complete its task. Processors can only communicate with their fathers or sons, if they exist. This type of parallelism is exploitable on almost any type of parallel machine, as the problem can be broken down into an arbitrary number of modules and communication requirements are negligible. Actual speedups would depend on memory access latency through the processor network and not interprocess communication. The algorithm shows a high tendency towards data parallelism in the lower levels of the quad tree with bottlenecking occurring at the high levels of the quad tree.

The order of complexity of the algorithm depends on the type of terrain and the error tolerance. This is because for low error tolerances, the algorithm is dominated by polygon comparisons along the boundary lines of sons, and for high tolerances, the algorithm is dominated by the merging of single polygons on each level. For the former, the time increases by a factor of 2 on each level. This is because the length of the mergeable boundaries

increases by a factor of 2 on each level. Since the number of processors decreases by a factor of 4, this results in a low bound of $O(\sqrt{n})$ where n is the number of processors at the base of the tree. For the high error tolerance/smooth terrain case, the amount of time spent on each level is the same (if the tree merges the terrain to one polygon), so the high bound on the order of the algorithm is simply the number of processors divided by the number of levels in the tree, or n/log n. This corresponds well to the values found in the parallel simulations. Speedup for most of the tests falls in between these two values.

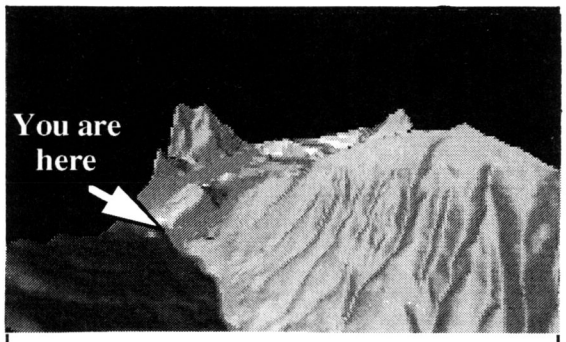


Figure 11. View of Banff

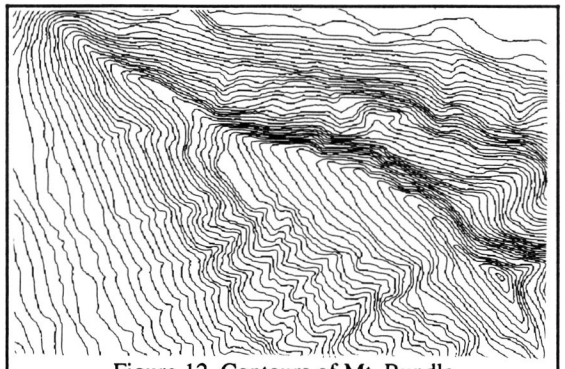


Figure 12. Contours of Mt. Rundle

3.3 Continuous resolution and critical line preservation

The tendency of the algorithm to preserve critical lines is shown in Figures 7 and 8. Both of these figures show Mount Timpanogos in Orem, Utah. Figure 7 uses 101,000 triangles, and Figure 8 uses 1,200. The ridgelines in both figures are nearly identical, while less important details on the face of the mountain have been lost in Figure 8. Continuous resolution is best demonstrated using video media. Figure 10 shows an interpolated triangular grid. Distortion of the rectangular polygons from the lower tolerance level is evident as they are transformed to the corresponding polygons in the higher tolerance level.

Figure 11 shows a view looking from the canyon west of Mt. Rundle towards Banff. Figure 12 shows part of the cleaned contours used to create the height grid of Banff. It should be noted that since the input data had 100-foot contour lines, no details smaller than 100 feet are visible in the rendering.

3.4 Execution time and hardware

A complex (rough) 256X256 terrain is polygonalized to 100 distinct levels of error tolerance in about 55 seconds on an HP750 workstation. Interpolating between two error tolerance levels takes about 12 seconds for an interpolation between 100,000 polygons and 40,000 polygons, and less than a second for interpolations where the lower level has less than 8,000 polygons. This time includes drawing the polygonal mesh to an X window. Rendering 100,000 polygons using StarBase takes 8 seconds on the workstation, which has no special graphics hardware.

4. Conclusions

This paper has described a new polygonalization procedure which makes essential use of a quad tree to construct a polygonal description of a discrete surface at multiple levels of detail. This procedure provides continuous resolution polygonalizations, is highly parallel, extracts and preserves critical lines automatically, allows rapid neighbor polygon searches, and provides easy access to any level of detail. This algorithm can also be used to automatically select the correct data points to preserve critical lines for triangulation algorithms. While automatic feature extraction has addressed this problem to some extent, this research presents a massively parallel critical feature extractor that extracts a set of points that can be triangulated in a highly parallel fashion.

The hierarchical structure generated when polygonalizing the height grid at each level of error tolerance possesses distinct advantages over the sequential Delaunay triangulation. The most important advantage is the ability to select and locate polygons in the structure using a tree search (log n) instead of a sequential search (n). Modifications to the Delaunay

algorithm which rely on hierarchical models [DEFLORIANI 89] overcome this limitation, but at the expense of non-parallel tree construction. In addition, continuous resolution capability requires a hierarchical structure (i.e., a polytree) to link and describe the changes from level to level. Triangle-based algorithms can also do this, except that the result is not as accessible in terms of specific locations on the height grid. Also, continuous resolution allows interpolation of an arbitrary level of detail from a discrete number of hierarchically coupled polygonalizations, which avoids computation and storage requirements associated with conventional polygonalizations.

The ability to have multiple levels of resolution on one display at the same time is a future research topic. Also of interest is the extension of this algorithm to three-dimensional anatomical structures in medical imaging, such as skulls.

Bibliography

S. Baronti, A. Casina, F. Lutti, et al, "Variable Pyramid Structures for Image Segmentation", *Computer Vision, Graphics, and Image Processing*, March 1990, 346-356.

Alan H. Barr and Brian Von Herzen, "Accurate Triangulations of Deformed Intersecting Surfaces", *Computer Graphics*, July 1987, 103-110.

William A. Barrett, "Morphological Contour Interpolation", GI 94, Banff, Canada

Peter J. Bert, Tsai-Hong Hong, and Azriel Rosenfield, "Segmentation and Estimation of Image Region Properties Through Cooperative Hierarchical Computation", *IEEE Transactions on Systems, Man, and Cybernetics*, Dec. 1981, 802-809.

Leila De Floriani, "A Pyramidal Data Structure for Triangle Based Surface Description", *IEEE Computer Graphics and Applications*, March 1989, 67-78.

Robert J. Fowler and James J. Little, "Automatic Extraction of Irregular Network Digital Terrain Models", *Computer Graphics*, August 1979, 199-207.

M. J. McCullagh, "Mini/Micro Display of Surface Mapping and Analysis Techniques", *Cartographica* 19:2 (Summer 1982), 136-144.

Sydney M. Petersen, William A. Barrett and Robert P. Burton, "A New Morphological Algorithm for Automated Interpolation of Height Grids from Contour Images", 1990 SPIE/SPSE Symposium on Electronic Imaging Science and Technology, Santa Clara, Feb. 1990.

Hanan Samet and Robert E. Webber, "Hierarchical Data Structures and Algorithms for Computer Graphics, Part I", *IEEE Computer Graphics and Applications*, May 1988, 48-68.

Lori L. Scarlatos, "A Refined Triangulation Hierarchy for Multiple Levels of Terrain Detail", Proceedings, IMAGE V Conference, Phoenix, Arizona, 19-22 June 1990.

Albin Tarvydas, "Terrain Approximation by Triangular Facets", ASP-ACSM Convention, Technical Papers, 1984 Vol. 1, 524-533.

This work was partially supported by a grant from IBM.