Right-Triangulated Irregular Networks

William Evans[*]
Department of Computer Science
University of Arizona

David Kirkpatrick[†]
Department of Computer Science
University of British Columbia

Gregg Townsend
Department of Computer Science
University of Arizona

Abstract

We describe a hierarchical data structure for representing a digital terrain (height field) which contains approximations of the terrain at different levels of detail. The approximations are based on triangulations of the underlying two-dimensional space using right-angled triangles. The methods we discuss permit a single approximation to have a varying level of approximation accuracy across the surface. Thus, for example, the area close to an observer may be represented with greater detail than areas which lie outside their field of view.

We discuss the application of this hierarchical data structure to the problem of interactive terrain visualization. We point out some of the advantages of this method in terms of memory usage and speed.

1 Introduction

In this paper, we describe a method for approximating a surface which is presented to us as a two-dimensional array of height values. We assume that the height value in location i, j of the array is the true height of the surface at location x_i, y_j where x_i and y_j are easily derived from the indices i and j. Thus we ignore questions of error in the input. For our purpose, the surface is defined by the values in the height array. Of course, this leaves most of the surface, those points with x, y coordinate not equal to x_i, y_j for some i, j, undefined. We make no assumptions about these intermediate points. We measure the goodness of our approximations only against the elevations in the array.

Such surface representation are common and can be quite large. For example, a 10 meter resolution 7.5 minute digital elevation model (DEM) of Mount Rainier, Washington [1] has $977 \times 1405 = 1,372,685$ data points. Combining several of these map sheets, which is necessary, for example, in accurate watershed calculation, can result in arbitrarily large data sets.

Computations involving such data sets are very expensive. Both memory and time used by the analysis algorithm can be prohibitively large. In such a case, we cannot use the entire surface representation. Either the surface must be subdivided into manageable pieces, which only works if the analysis can be localized to these pieces, or we must use an approximation to the surface in place of the original. Of course, the approximation should be constructed so that the answers we obtain in analyzing it approximate the answers we would obtain for the true surface.

Surface approximation of regular gridded DEMs typically possesses one of two forms. Either the approximation is itself a regular gridded DEM, a regular *sub-grid* of the original data, or it is a

[*]Research supported by NSERC Canada International Fellowship and NSF Grant CDA-9500991.
[†]Research supported by NSERC Research Grant A3583.

triangulated irregular network (TIN), a possibly irregular subset of the original data points whose projection into the x, y-plane is triangulated.

The approximation form we describe in this paper lies somewhere between the regular sub-grid and the irregular TIN. It is more regular than the TIN and more flexible than the sub-grid. Like the TIN, it is a triangulated subset of the original data points, but, unlike the TIN, the subset is not arbitrary. The subset is such that all triangles are right angled and isosceles. Consequently, we call this form of approximation a right-triangulated irregular network (RTIN).

The main benefit of the RTIN approximation is not in providing a single approximation to a surface, but rather in providing a framework of many approximations at varying levels of detail.

A given approximation may be well suited to one type of approximation but not another. An approximation that represents ridge lines very accurately may not be the best approximation for determining wildlife habitat. In order to form an appropriate approximation to the surface, we need to know for what purpose the approximation is intended. It is time consuming and often impractical to create and store an approximation for each intended application. Often a better solution is to devise a general framework for approximation that can yield approximations tailored to particular applications. The problem then becomes calculating the appropriate approximation from the framework. Extracting an approximation from an existing framework has the potential advantage of providing tailor-made approximations much more rapidly than constructing a suitable approximation from scratch. Of course, this works well only if the framework can provide an appropriate approximation quickly. This is critically important when many different approximations are required in rapid succession.

The framework we propose in this paper is a hierarchy of approximations. It is a hierarchy in that the framework contains a range of representations, from coarse to fine. However, the representations are not segregated. The hierarchy can provide a single approximation that mixes coarse- and fine-level representations. This allows us to obtain high levels of detail in certain regions of the surface without forcing us to represent the entire surface at this high resolution. For example, in order to accurately approximate elevation, mountainous regions may require much finer detail than flat plains.

An additional benefit to having the entire hierarchy of approximations is that it can make tasks such as point location faster than they would be if one had only a single approximation.

The disadvantage to keeping a hierarchy of approximations is that it requires more storage than a single approximation. One of the main goals of this work is to demonstrate a practical and space efficient approximation hierarchy. Our implementation of the RTIN hierarchy requires less than 14 bytes per original data point. This includes two bytes for the height value itself[1]. Though seven times more expensive than the original data set, the hierarchy provides an extensive range of approximations.

To illustrate the utility of the RTIN hierarchy, we describe its implementation for interactive visualization of a surface. The goal of interactive visualization is to show what a user would see as they "fly" or "walk" over the surface. The problem is that the surface is too detailed to render at full resolution at a reasonable speed. Thus, the surface must be approximated in order to reduce the time needed to update the display. A single approximation, independent of the eye position, could be used, but the resulting image may be unacceptably coarse in areas close to the eye. Our approach is to adapt the approximation to the location of the eye so that regions close to the viewer are at high resolution while distant or non-visible regions are more coarsely represented.

The demand for varying levels of detail within one approximation must be answered carefully. Areas of high detail must match with adjacent areas of low detail in such a way that the surface

[1]Our input is U.S. Geological Survey DEM data which uses two bytes to represent an elevation.

remains “continuous”. Discontinuities are very noticeable in interactive applications.

Interactive visualization is a good test of our hierarchy since it requires many of the desired properties: we must be able to choose an appropriate approximation rapidly; the approximation must allow different levels of detail at different locations on the surface; and the surface must be a continuous surface after interpolation (gaps in the surface can be glaringly obvious).

The focus of this paper will be on describing the RTIN hierarchy and the data structure used to implement it. Later sections of the paper will detail the performance of the hierarchy both as a solution to the interactive visualization problem (section 7.2) and as a way to represent a single (non-hierarchical) surface approximation (section 7.1).

We start by reviewing a portion of the large body of work that has been done on surface approximation. We then review the two popular surface representations, the sub-grid and the TIN, and describe hierarchies based on these representations. This sets the stage for our discussion of the RTIN hierarchy of approximations (section 5).

2 Related Work

A great deal of work has been done on the approximation of surfaces, and, in fact, on the particular problem of *multi-resolution surface modeling* of which hierarchies are one type. A recent survey by De Floriani, Marzano, and Puppo discusses many of the advances in this field [2]. This survey does not refer to a large body of more recent work that has appeared in SIGGRAPH96 [3, 4] and EuroGraphics96, both of which had sessions on multi-resolution surface modeling. A recent tutorial by Puppo and Scopigno [5] covers this more recent work as well as a wide range of surface simplification techniques.

The most relevant work to our approach that has appeared in the surface approximation literature is the work by Lindstrom et al. [6] and Puppo [7, 8]. Both independently discovered hierarchies similar to the one discussed in this paper. We discuss and compare their approaches to ours in section 8.

Another source of related work comes from the study of general lattice structures. Bell et al. [9] cite work on crystal lattice structures by Šubnikov in 1916 [10] and Laves [11] in 1931 that defined various planar partitions including the one upon which our hierarchy is based. These partitions have been studied as a means of addressing spatial regions with application to image encoding [12, 13] and adaptive mesh generation for finite element solvers in two and higher dimensions [14, 15]. Hebert appears to be the first to describe non-uniform versions of the particular partition discussed in this paper, though the concept is closely related to the quad-tree data structure discussed by Samet [16] and the the *restricted quadtree* of Von Herzen and Barr [17].

3 Sub-grid and TIN

Approximations of height data typically come in one of two types: sub-grid or TIN. A sub-grid is of the same form as the input array of height values. It is an array of height values at regularly spaced x and y coordinates. Since the sample points are regularly spaced, it is possible to calculate the true x, y coordinates of a height value from its array indices. Thus, the only storage needed is for the height or z values of the sample points. To obtain the height at x, y coordinates not in the sample set, one uses some form of interpolation. In fact, what is often done is that the four sample points forming the smallest square containing the x, y position are used to determine the height at x, y via bilinear interpolation. The regular spacing of the sample points makes it easy to determine the closest sample points to a given x, y position. Alternatively, one might choose to use linear interpolation based on a triangulation of this smallest square (arbitrarily picking one of the square's two diagonals to split it into two triangles).

The sample points of a TIN (Triangulated Irregular Network) are an arbitrary subset of the original data points. In order to specify this subset, the x, y coordinates (i.e. the indices in the input array) of each sample point in the TIN must be stored explicitly, in addition to the z value. The sample points, without their height component, are triangulated (typically using a Delaunay triangulation) and linear interpolation is used to obtain the height at an arbitrary point (x, y) from the heights of the vertices of the triangle which contains the point.

The advantage of the TIN is that the sampling can be nonuniform: large featureless regions can be sampled at a coarser resolution than regions with a great deal of variation. The disadvantage is that a TIN requires more storage for the same number of sample points; a 3:1 ratio if x, y, and z values require the same precision. The storage disadvantage becomes worse if one requires adjacency information for the triangulation; if, for example, one needs the ability to traverse the surface approximation from triangle to adjacent triangle.

The factor of three difference in storage requirements has led to an unfavorable view of TINs. Kumler [18] argues that to achieve a certain degree of accuracy in the representation of a height field, sub-grids require less storage space than TINs. However, he then states, "I did not expect these results. My intuition led me to believe, and I continue to believe, that the irregular TIN model is, in general, a more efficient method for representing an irregular natural surface." He then goes on to explain his conviction by claiming that methods to construct more efficient TINs will be developed in the future. Our results comparing error measures to space usage for sub-grids and TINs may shed some light on this issue (section 7.1).

4 Sub-grid and TIN hierarchies

Both sub-grid and TIN approximations may be organized into hierarchies. A hierarchy of sub-grid approximations is obtained by varying the spacing of the regularly sampled grid points. For example, one level of the hierarchy may contain every other sample point from the input array while a coarser level may contain every fourth sample point. Typically, each level of the hierarchy is a regularly subsampled approximation of the next finer level. Thus, the amount of storage needed for the entire hierarchy is just the amount needed for the finest level of approximation. The price of such small storage is the rigidity and uniformity of the sampling pattern.

A TIN hierarchy allows varying levels of detail within an approximation at the expense of more storage. There are several hierarchies based on irregular triangulations [19, 20]. One may roughly divide them into those that preserve the boundaries of coarse triangulations in going to finer triangulations, and those that do not. In the first case, the hierarchy is organized in a tree structure; each node represents a region, and the children of a node represent the regions into which the node's region is partitioned in going from one level to the next finer. Refinement is typically performed by choosing within each region a point from the original input data, adding it to the set of sampled points, and retriangulating (maintaining the previous boundaries). Such a scheme tends to create long, thin triangles which may provide a less accurate approximation than shorter-edged, larger-angled triangles. On the other hand, any level of detail can be achieved in one area of the triangulation without affecting other areas. In terms of the tree structure, any subtree containing the root represents a valid surface approximation.

In the second case, one can choose to re-triangulate at each level using, for example, Delaunay triangulation. In this case, it is not in general possible to refine a particular region independent of other regions, i.e. combining pieces from different levels of the hierarchy is difficult. DeBerg and Dobrindt recently showed how a hierarchy of Delaunay triangulations which allows local refinement can be constructed. This is accomplished using a more complicated structure than the simple tree used in the first case. DeBerg and Dobrindt cite memory usage of 132 bytes per data point[2].

Our goal is to provide a hierarchy of surface approximations which is a compromise between the sub-grid and the TIN approximations. We hope that a more regular "grid-like" TIN will combine the sub-grid's advantage of smaller space per vertex with the TIN's advantage of local adaptability to terrain.

5 RTIN Hierarchy

Our hierarchical surface representation contains partitions, called RTIN partitions, of the two dimensional square into right-angled, isosceles triangles. Such partitions have been studied in relation to crystal lattices [9], image encoding [12], and, in fact, geographic visualization systems [6, 7, 8]. The partitions contained in the hierarchy are defined inductively as follows. The partition composed of two triangles formed by dividing the square from northwest to southeast corner is the coarsest partition in the hierarchy (i.e. the one containing the largest triangles). From a partition in the hierarchy, a new, more detailed partition may be formed by *splitting* any one of the *non-terminal* triangles in the partition, where a non-terminal triangle is one which is larger than some fixed threshold depending on the resolution of the underlying height field. A triangle T is split by adding an edge from its right-angled vertex to the midpoint of its hypotenuse. The resulting partition may be *non-triangular* meaning that the new vertex introduced at the midpoint of the hypotenuse may lie on the border of a triangle R (T's neighbor across the hypotenuse) turning R into a quadrilateral. This represents a problem when using the partition to create a continuous surface via linear interpolation. Each vertex in the partition has a height value defined by the height field. Since R is a quadrilateral, linear interpolation over R is not well-defined, and performing linear interpolation using R's original three vertices may cause gaps in the surface. To avoid this problem, we continue or *propagate* the split into R. If R is larger than T then the hypotenuse of T forms a leg of R and we first split R (perhaps propagating this split) and then split the resulting triangle which shares its hypotenuse with T. If R is the same size as T we need only split R (no further propagation is necessary). See figure 1.

Propagation means that a single split operation can cause more than one triangle to split. However, splitting a triangle T cannot cause a triangle smaller than T to be split. Thus, only a finite number of triangles will be split in a single split operation, and the operation is guaranteed to terminate. In fact, at most two triangles of each size equal to or larger than T and no triangles smaller than T will split as a result of splitting T.

The most detailed partition in the hierarchy is one in which no triangle may be split, i.e. all triangles are terminal. The most detailed partition is *uniform*: every triangle is the same size. The vertices of the partition form a grid of size $(2^k + 1) \times (2^k + 1)$ for some integer k. The height of the (i, j)th grid point is the height of the (i, j)th sample point in the height array. The integer k (or, equivalently, the threshold defining the terminal triangle) is chosen so that the grid is large enough to contain the height array.

If extended to cover the plane, such a uniform partition is called a $[4.8^2]$ Laves net named by Bell, Diaz, Holroyd, and Jackson [9] after F. Laves. The name lists the degree of the vertices of

[2]Memory usage can be reduced by increasing the difference in detail between levels, but the hierarchy is then less capable of matching the exact resolution requirements of a desired approximation.

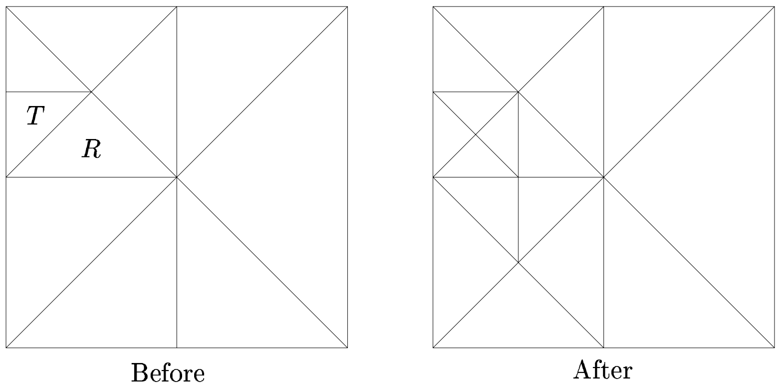

Figure 1: Splitting triangle T.

the atomic polygon within the partition, in cyclic order around the polygon. In this case, the right angled vertex has degree 4, and the two other vertices both have degree 8.

Other Laves nets which may be used as the basis for single shape hierarchies are the square $[4^4]$, the equilateral triangle $[6^3]$, and the 30-60 right triangle [4.6.12]. See figure 2. These nets all share the crucial property that they are infinitely refinable using similar polygons: a given polygon in any net can itself be partitioned using polygons of the same shape.

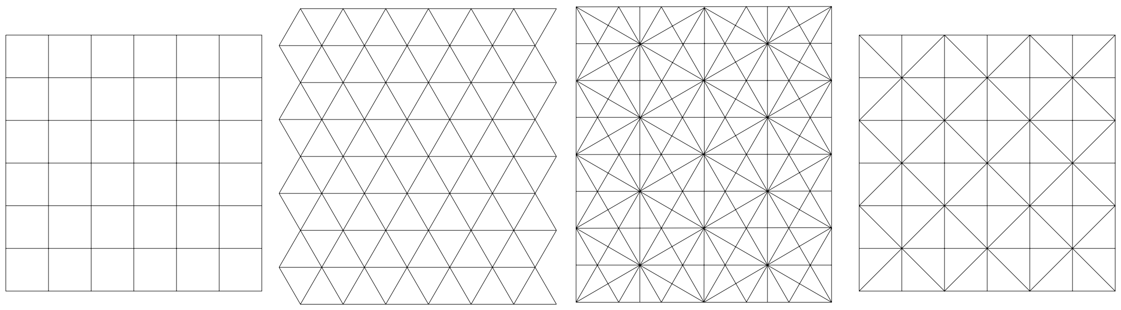

Figure 2: Pieces of Laves nets of type $[4^4]$, $[6^3]$, [4.6.12], and $[4.8^2]$.

Unlike a Laves net, a partition may be non-uniform. Only certain Laves nets can form the basis of non-uniform partitions. For partitions based on the square or equilateral triangle net, refining one region forces all other regions in the partition to be refined. This refining is forced in order to maintain the basic shape of the regions in the partition. For example, refining one square in a square net introduces new vertices on the edges of adjacent squares which forces these squares (now pentagons) to be refined, which forces further refinement until the partition is uniform. The equilateral triangle net behaves in a similar manner. Thus these nets, without special compensation for non-similar regions, cannot be the basis of a non-uniform partition. The [4.6.12] (30-60 right triangle) partition and the $[4.8^2]$ (isosceles right triangle) partition do not suffer this limitation. As mentioned earlier, refining (or splitting) a triangle affects at most two triangles of each size in the $[4.8^2]$ partition. In the [4.6.12] partition, the number of triangles affected of each size equal to or larger than the triangle initially split is at most 12. See figure 3.

Three properties favor the $[4.8^2]$ partition for hierarchical representation of surface approximations. First, it is a triangular subdivision which means that height values in the interior of the

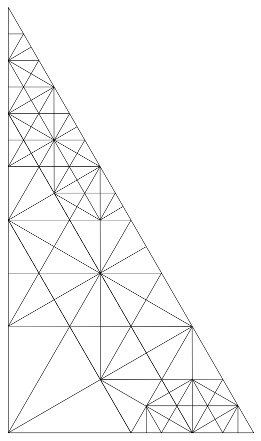

Figure 3: Non-uniform partition based on the [4.6.12] Laves net.

triangle can be linearly interpolated from the height values at the vertices of the triangle in an unambiguous way. The benefits of linear interpolation over other methods of interpolation (such as bilinear or inverse-distance weighting) are speed and simplicity. For 3D visualization, linear interpolation is supported in hardware on some machines, making it the only reasonable interpolation scheme for large numbers of triangles.

Second, the vertices of a uniform $[4.8^2]$ partition are equally spaced grid points, in contrast to a [4.6.12] partition. Thus, each vertex represents a sample point from the original data set, and every point from the data set is included in the most detailed partition.

Third, one area of the partition can be refined, while maintaining the "no gap" or surface property, without affecting a large number of regions. Refining or splitting a triangle affects only a small number of other triangles, in contrast to the $[4^4]$ or $[6^3]$ partitions.

5.1 Data Structure

The previous section introduced RTIN partitions. We now discuss how these partitions may be represented in a data structure both individually and collectively as a hierarchy. The general idea is suggested by the definition of the RTIN partition: since each triangle may be divided into two similar pieces, a partition is represented by a binary tree. In particular, each (triangular) region in the partition is represented by a leaf in the binary tree that represents the partition.

The root of the tree represents the initial square (the only case in which the region associated with a node is not a right triangle). The children of the root correspond to the regions obtained by dividing the square along its northwest-southeast diagonal. The left (right) child corresponds to the triangle containing the southwest (northeast) corner. In general, the children of a node with triangular region T correspond to the triangles obtained by "splitting" T from the midpoint of its hypotenuse to its right-angled vertex. The left (right) child is the triangle to the left (right) of this split (looking from the midpoint to the vertex). See figure 4.

This establishes a labeling scheme for the triangular regions in a RTIN partition. The label of a region is a description of the path from the root to its corresponding node in the binary tree. The path description is the concatenation of the labels of the edges on the path from the root to the node, where an edge is labeled 0 (1) if it leads to a left (right) child. An example of a RTIN partition and its representative binary tree are shown in figure 5.

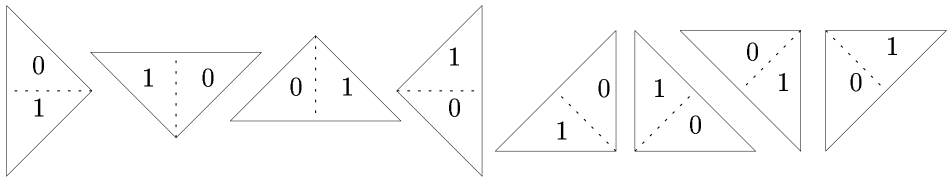

Figure 4: The possible triangle orientations and their left and right children. Left and right are denoted by 0 and 1 respectively.

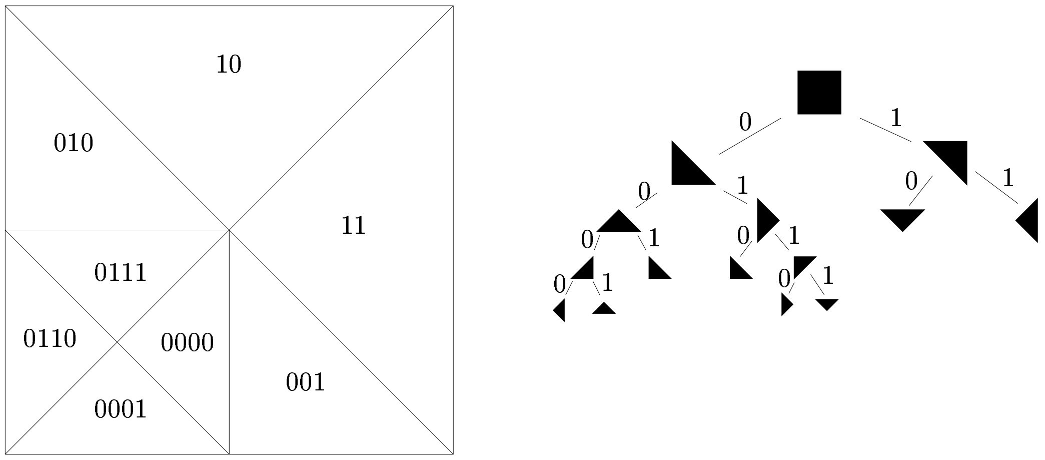

Figure 5: The binary tree representation of a RTIN partition.

Any *single* partition corresponds, as in figure 5, to a binary tree. Thus a RTIN surface approximation can be represented by a binary tree and the height of each of the triangle vertices from the input elevation model. The structure of the tree allows us to calculate the (x, y) coordinates of the vertices of a triangle from its label in the tree. This is the advantage of the RTIN structure over a general TIN. Neither the coordinates nor the label need to be explicitly stored. Contrast this with the general TIN which must store the (x, y) coordinates (or indices) of the vertices of the triangles it contains. In this way, the RTIN resembles the regularly sub-sampled grid. In the sub-grid, the x,y coordinates of a point can be calculated from the point's indices in the array of height values. In the RTIN, the x,y coordinates of the vertices of a triangle can be calculated from the triangle's label. Both the point indices in the sub-grid and the triangle label in the RTIN are *addresses* into the structure. They are not stored in the structure. Unlike the regular sub-grid, the RTIN allows nonuniform sampling; some regions may be more densely subsamples than others. In this way the RTIN resembles the general TIN.

Determining the coordinates of the three vertices of a triangle from its label is straightforward. The label describes a path in the binary tree representing the surface. At each step in this path, as one descends from the root, one can construct the vertices of the left or right child triangle from the vertices of the parent. If $\Delta(v_1, v_2, v_3)$ is the parent triangle (vertices are listed in counter-clockwise order with v_3 the right-angled vertex) then the left child is $\Delta(v_3, v_1, m)$ while the right child is $\Delta(v_2, v_3, m)$ where (m_x, m_y) (the x,y coordinates of m) are the x,y coordinates of the midpoint of the line segment v_1v_2, and the z coordinate (i.e. height value) of m is obtained from the input array location (m_x, m_y). We assume that the x, y coordinates of the vertices are integers and represent indices into the input array. If not, they can be scaled and rounded to the appropriate integral range.

A straightforward implementation of this binary tree structure for a *single* surface approximation is inefficient in its use of space. Each node requires two pointers to its children and the height of its vertices must be stored as well[3]. This, along with the fact that there are twice as many nodes in the tree as triangles in the approximating surface, makes the RTIN method of single surface approximation more space intensive than a TIN representation. See section 7.1 for a more detailed examination of space usage in the case of a single surface approximation.

The advantages of the RTIN become more apparent when one wishes to represent a *hierarchy* of surface approximations rather than just a single approximation. In some sense, the binary tree representation of a RTIN partition is tailor-made for hierarchies since, if it represents a partition at a particular level of detail, then it contains the representation of every coarser approximation. Thus our hierarchy is simply the binary tree that corresponds to the most detailed RTIN partition for the input elevation model.

Of course, to obtain a surface approximation from the binary tree, we also need the height values of the vertices of all triangles in the approximation. For this we store the original input array of elevations.

For the price of this one RTIN surface approximation (albeit the most expensive), we obtain a representation of *all* RTIN surface approximations. Methods for extracting a particular surface approximation from this hierarchy are described in section 6.

5.2 Reducing Space Usage

This section describes issues concerning the space used by the hierarchy and how that space may be reduced.

If the original input was a $2^k+1 \times 2^k+1$ array, the binary tree representing the hierarchy is the complete binary tree of depth $2k+1$. Each leaf of the tree represents a triangle that is half of the input grid square. In this case, it is much more space efficient to store the binary tree as an array where the label of a node, treated as the binary representation of an integer, determines the node's location in the array. The problem of having, for example, labels 0 and 00 both representing the same integer can easily be solved by prepending a 1 to the node label to form the array index. The addressing scheme obtained in this way corresponds to the typical implicit array representation of a binary heap [21]. By using the array representation, we eliminate child pointer storage.

If the input does not have size $2^k+1 \times 2^k+1$ but rather size $w \times h$ (where w and h are less than 2^k+1 but either w or h are greater than $2^{k-1}+1$) then a complete binary tree of depth $2k+1$ contains some nodes that do not correspond to triangles with vertices that are grid locations. The triangles represented by these leaves do not "cover" the input array (i.e. their projection into the x, y-plane does not intersect the range of the indices of the input array). They are invalid. A more complicated indexing scheme than the implicit array representation can be used in this case. The basic idea of this scheme is to follow the implicit array representation but to calculate which of the nodes in the binary tree represent invalid regions, and to offset the implicit array index of a node by the number of invalid regions whose implicit array index precedes it. The space requirement is then reduced from being proportional to $(2^k+1)^2$ to being proportional to wh.

Since these space saving mechanisms eliminate the need for a node to store pointers to its

[3]One possible location in which to store that height of a vertex is in the node whose hypotenuse midpoint is that vertex. This means that the height value of a particular data point may be stored twice; once for each triangle that has that point as the midpoint of its hypotenuse.

children, one might well ask what information is stored in the tree, since the pointer structure and the height values were the only pieces of information required to represent a single surface approximation.

The only vital piece of information that must be stored in the nodes of the hierarchy is an indication of whether or not the triangle is part of the approximation surface. In the single surface approximation, the fact that a node was a leaf of the binary tree indicated that it represented a region in the approximation. In the hierarchy, which is used to represent many different approximations, this can be accomplished using a single bit per node which indicates whether or not the node's triangle is a part of the approximation. The total storage of a $(2^k+1)\times(2^k+1)$ image is the amount used for the array of height values ($(2^k+1)\times(2^k+1)$ words) plus one bit for every node in the tree ($2^{2k+1}-1$ bits).

This representation, however, is too sparse to be of much use for any algorithm which operates on the surface approximation. One might reasonably demand that each node contain some representation of the error of its triangle[4] in order to determine how well the approximation fits the true surface, and in order to calculate appropriate single approximations from the hierarchy (see section 6).

One may also demand the ability to "walk" from one triangle of the surface to a neighboring triangle of the surface. That is, given the address of a triangle, calculate the addresses of the (at most) three triangles that share an edge with it. We describe in the next section how this may be accomplished using an additional three bits of storage per triangle.

5.3 Neighbor Calculation

Often the algorithms one needs to execute on the approximation surface require the ability to traverse the surface from triangle to adjacent triangle. For example, one algorithm for calculating the watershed of a stream performs a hill-climbing operation which follows a path of steepest ascent from triangle to adjacent triangle [22]. In order to accomplish this, the algorithm needs to determine the neighbors of any given triangle. In particular, one needs a function that, given the label of a triangle, can calculate the label of its adjacent triangles.

In order to describe this function, let us number the vertices of a triangle in counter-clockwise order from 1 to 3 so that vertex 3 is the right-angled vertex. Define the *i-neighbor* of a triangle as the neighbor that does not share the triangle's vertex i (see figure 6). The *same-size i-neighbor*

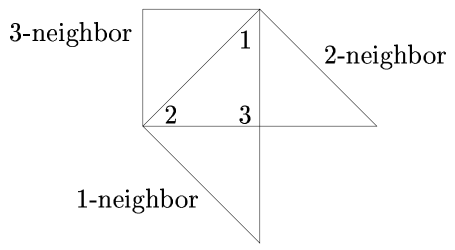

Figure 6: The i-neighbors of a triangle. In this case, all neighbors are the same size as the triangle. The numbers within the inner triangle are the vertex numbers of the inner triangle.

of a triangle t is the triangle's i-neighbor in the uniform partition that contains t. The same-size

[4]One possible error measure is the maximum vertical distance between a point whose x,y projection lies within the x,y projection of the triangle and the triangle's surface

neighbor may or may not be a part of the approximation surface. The function which finds the i-neighbor of a triangle first finds the same-size i-neighbor of the triangle and then uses some extra information stored with the triangle to determine the true i-neighbor. The extra information is a bit which says whether or not the i-neighbor is the same size as the triangle or not. If the bit indicates that the i-neighbor is not the same size as the triangle then the i-neighbor must be smaller if $i = 1, 2$ or larger if $i = 3$. Alternatively, one may avoid the extra 3 bits of storage by examining the surface bit of the same-size neighbor and its parent (or child).

The following recursive equations calculate the labels of the same-size neighbors of a triangle specified by its label. $N_i(t)$ is the same-size i-neighbor of t. The symbol λ is the (empty) label of the root. The symbol $\emptyset$ represents "no neighbor". The function $\circ$ is concatenation with the understanding that $\emptyset \circ 0 = \emptyset \circ 1 = \emptyset$.

$$\begin{array}{rrr}
N_1(\lambda) = N_1(0) = N_1(1) = \emptyset & N_2(\lambda) = N_2(0) = N_2(1) = \emptyset & N_3(\lambda) = \emptyset \\
N_1(p0) = N_3(p) \circ 1 & N_2(p0) = p \circ 1 & N_3(0) = 1 \\
N_1(p1) = p \circ 0 & N_2(p1) = N_3(p) \circ 0 & N_3(1) = 0 \\
 & & N_3(p0) = N_2(p) \circ 1 \\
 & & N_3(p1) = N_1(p) \circ 0
\end{array}$$

An example of the use of these equations is given in figure 7.

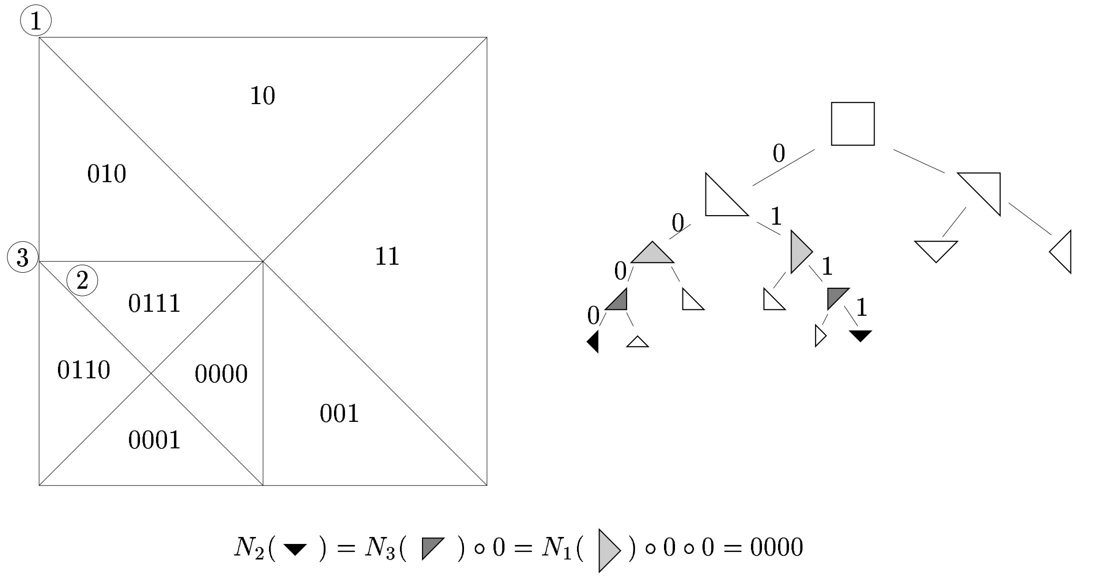

Figure 7: Use of the recursive neighbor equations to calculate the 2-neighbor of 0111: $N_2(0111) = N_3(011) \circ 0 = N_1(01) \circ 0 \circ 0 = 0000$.

A recursive algorithm based on these equations is fast enough in nearly all situations. Nonetheless, it may be of interest to note that these calculations may be performed using a small number (unrelated to the length of the label) of arithmetic and bitwise logical operations, provided the label is short enough to fit in a computer word (see the C-code in figure 8). The code works by first reducing the problem to calculating $N_3(t)$; the 1-neighbor or 2-neighbor of a triangle t is simply

```
int neighbor[3][4] = {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 2, 1}};
#define SIEVE0 0xAAAAAAAAAAAAAAAA
#define SIEVE1 0x5555555555555555

int sameSizeNbr(int t, int i)
/* return the label of the same size i-neighbor of t. */
{
    unsigned int a,b,c,n;

    if (t <= 3) return neighbor[i-1][t];
    if (i == 1) t = (t << 1) | 1;  /* reduce to 3-neighbor calculation */
    else if (i == 2) t = (t << 1); /* reduce to 3-neighbor calculation */
    c = ((t << 1) ^ t) & SIEVE0;
    a = c | (c << 1) | 1;
    b = (a + 1) ^ a;        /* most of the work is done by this addition */
    if (b < (t>>1)) {
        n = t ^ b;
    } else if (((t & SIEVE0) << 1) > t) {
        if (((t & SIEVE1) << 2) > t) {
            n = t ^ (b>>1);
        } else {
            n = t ^ (b>>3);
        }
    } else {
        return 0;         /* 0 means ``no neighbor'' */
    }
    if (i != 3) return (n >> 1);
    else return n;
}
```

Figure 8: C-code for fast triangle neighbor calculation.

the parent of the 3-neighbor of a child of t. From the recursive equations, if $t = uvw$ where v is either 00 or 11 and w is in the regular language $\{01, 10\}^*$ then $N_3(t) = u\overline{vw}$ where $\overline{x}$ is the bitwise complement of x. This is the property exploited by the sample C-code. Complementation of vw is accomplished by a single addition.

6 On-the-fly Surface Approximation

Using the RTIN hierarchy defined in the previous section, we can create an algorithm that quickly constructs a surface approximation. Since the application we have in mind is interactive visualization, the speed at which the approximation can be constructed is of prime importance. The construction time together with the rendering time determines the number of scenes that can be drawn per second.

There are, essentially, two approaches to constructing an approximation from a hierarchy: bottom-up and top-down. Bottom-up starts with the finest, or most accurate, approximation and progressively relaxes it – letting a large triangle, for example, replace smaller triangles if the large triangle well-approximates the smaller ones. Top-down starts with the coarsest approximation and progressively refines it. Bottom-up works well in cases where the approximation is close to the finest approximation, but it must perform many relaxations in order to reach a coarse approximation. Top-down quickly reaches coarse approximations, but may waste time refining very coarse approximations if the desired approximation is very fine.

We use a top-down approach because the amount of work done to construct an approximation in a top-down fashion is proportional to the number of triangles in the approximation. Thus, reducing the number of triangles in the approximation speeds up the calculation of the approximation, as well as decreasing the time to draw the approximation. This assumes that the time to draw an approximation decreases as the number of triangles in the approximation decreases. This assumption is supported by our experimental results (section 7.2).

Given our choice of top-down method, we must decide how to choose the triangles to refine. Again, there are essentially two choices: *thresholding* or *prioritized refinement* (though many variations of these two themes are possible). Prioritized refinement chooses to refine the approximation in the most poorly approximated parts of the surface until the approximation contains a certain number of triangles. The single approximation results presented in section 7.1 use a version prioritized refinement that is explained there in detail. On the other hand, thresholding insists that the approximation everywhere satisfies some constraint. The constraint may be as simple as that every point in the input data lies within ϵ of the approximation, or it may be a more complicated constraint based on the eye position and a desired visual fidelity. In any case, the number of triangles produced in the approximation is not as tightly controlled as in prioritized refinement. A surface may require few triangles to achieve the desired overall fidelity in certain cases, while requiring many in other cases. Certainly, by decreasing the overall requirement, the number of triangles needed in the approximation decreases, but the control is not as precise as in the previous case.

Either method may be used to achieve surface approximations. Here we outline a method based on thresholding that is used in our implementation of an interactive visualization system.

An initial preprocessing phase allocates to each triangle a measure of how accurately it approximates the surface – the triangle's error. In our implementation this is the maximum over points "covered" by the triangle of the vertical distance from the point to the triangle. The preprocessing phase also initializes the RTIN hierarchy. After its completion, a triangle's error need never be recalculated. Other triangle specific information may be calculated and stored in the hierarchy at this time, for instance, the triangle's surface normal, its display properties, etc. This increases the space requirements of the hierarchy but may decrease the display time.

The heart of the algorithm is an augmented depth-first search of the hierarchy that updates the surface approximation in response to movements of the eye position. The depth-first search starts with the approximation being the two triangles represented by the two children of the root of the hierarchy (the binary tree representing the finest RTIN partition). It visits each child of the root. In general, if the search is visiting a node and the triangle that the node represents is not sufficiently close to the surface, then the depth-first search splits the triangle (which may cause many triangles to split recursively in order to avoid gaps in the surface) and then recursively visits the children of the node. The recursion caused by a triangle split may split triangles whose nodes have not yet been visited by the depth-first search. If the depth-first search visits a node whose triangle has been previously split, it recursively visits the node's children.

The decision of whether a triangle is "sufficiently close to the surface" is based on both the error of the triangle and the distance of the triangle from the eye position. In general, one may choose an inverse-distance weighting of the triangle's error, the area that the projection of the triangle

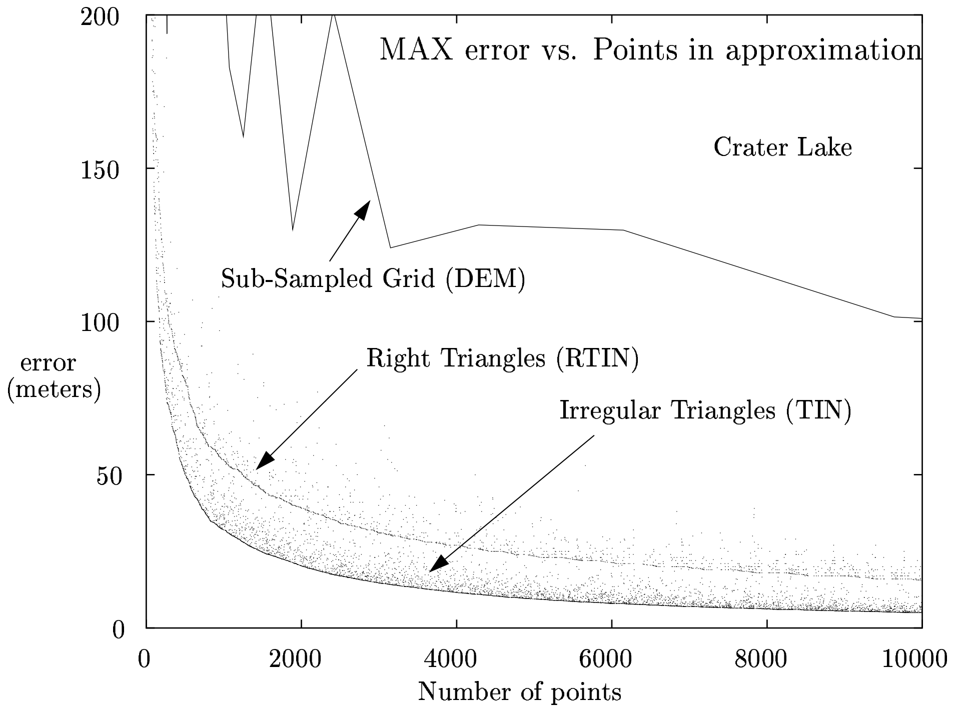

Figure 9: Maximum error versus number of points in approximation. Crater Lake data set.

occupies on the display screen, or other measures. (See deBerg and Dobrindt [20] and Lindstrom et al. [6] for examples.) One obvious common criteria is that triangles at the finest level of detail should not be refined.

The precise function used to determine if a triangle is "bad" is not our main focus. We present results in section 7.2 that call a triangle bad if the inverse weighting of its error is greater than some threshold; or if it is large and close to the eye position, e.g., within 700 meters of the eye position, any triangle that is not at the finest level of detail is bad. A more appropriate measure for interactive visualization might also include some measure of the visibility of the triangle. For instance, triangles outside the field of view, no matter how poorly they approximate the surface, are not seen by the observer and, hence, should not be bad.

Once the depth-first search creates the approximation, a second depth-first search traverses the hierarchy and draws the triangles that form the approximation.

7 Experimental results

7.1 Single surface performance

To construct a single surface approximation, we use a procedure similar to the general TIN construction method of Fowler and Little from 1979 [23]. The procedure is iterative and greedy. In each iteration, the point least well represented by the current approximation is added to the set of vertices, and the x,y projection of the set is re-triangulated (using a Delaunay triangulation) to form the next approximation. In our case, we do not have the flexibility of adding an arbitrary point to the current approximation. We can refine an approximation only by splitting a triangle that is in the approximation. However, we can, in spirit, mimic this greedy approach, by splitting the

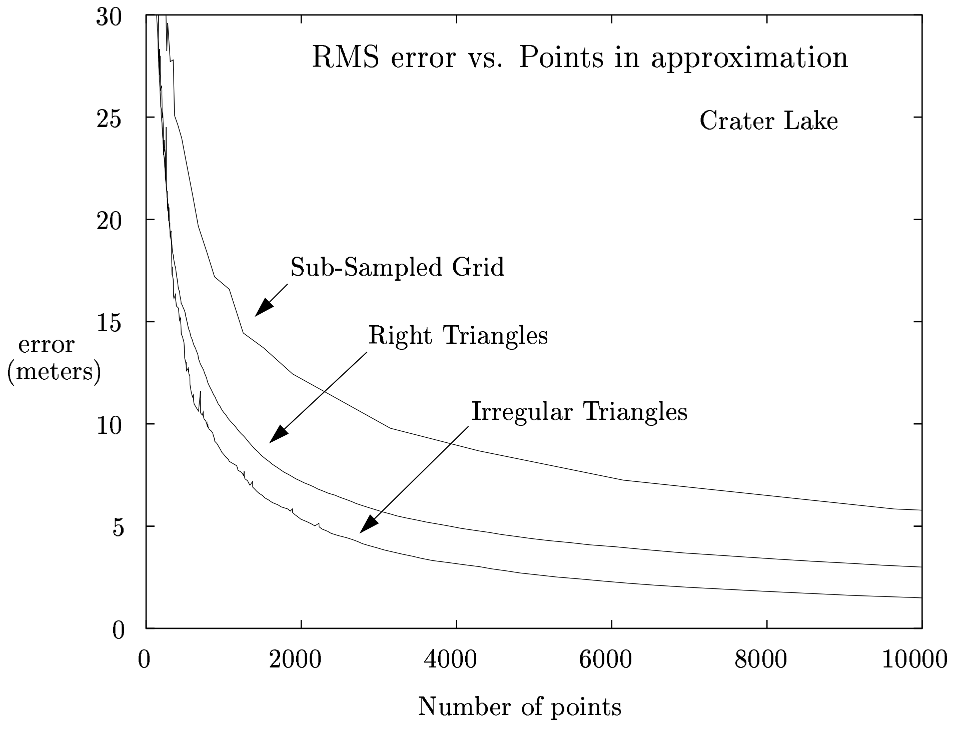

Figure 10: Root mean squared error versus number of points. Crater Lake data set.

triangle that *contains* the least well approximated point, perhaps causing other triangles to split. To implement this, we maintain a priority queue of the triangles in the approximation prioritized by their error and incrementally split the worst triangle.

For both TIN and RTIN construction, the procedure attempts to reduce the maximum error in the surface approximation at each iteration. It targets the point which has the current worst error, or the triangle which contains this point in the case of RTINs. If, however, instead of desiring a small maximum error, one desires a small RMS (root mean squared) error, then the analogous procedure is to find the triangle with the worst RMS error or the point whose deletion most decreases the RMS error. For TINs, finding the point that most decreases the RMS error is prohibitively expensive. However, for RTINs, this greedy heuristic is rather simple to adopt: at each iteration the triangle with the maximum RMS error is chosen to be split.

As one would expect, the maximum error in the surface decreases more rapidly as a function of the number of points in the approximation when we allow general triangles in the partition (the TIN approximation) as opposed to the more constrained RTIN approximation. A sub-sampled DEM is the worst performer under this measure (figure 9). Note that we used linear interpolation to calculate the surface in each case. The plots of the RTIN and TIN errors are scatter-plots. The points cluster so densely that they present the illusion of smooth curves. The plots show that, occasionally, adding a point to an approximation *increases* the maximum error. This is a result of using a non-optimal algorithm to calculate the RTIN and TIN approximations.

We obtain a similar result when we consider the RMS error as a function of the number of points in the approximation. In this case, the RTIN algorithm chooses to split the triangle which has the largest RMS error (figure 10).

Even when we consider the maximum error as a function of the amount of memory required

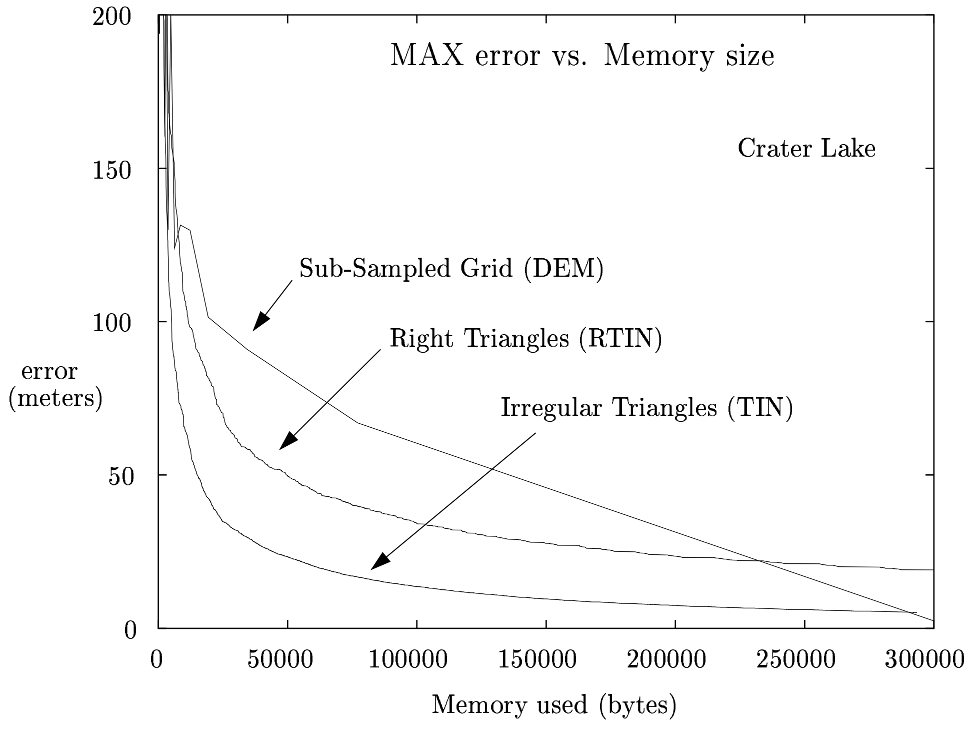

Figure 11: Maximum error versus memory usage. Crater Lake data set.

to store the approximations, the irregular triangulation or TIN still outperforms both the RTIN approximation and the gridded DEM approximation (figure 11). We count only the space for the approximation itself, and we assume that the coordinate values can be stored in two bytes.

In the case of the TIN, the amount of memory used is approximately 30 bytes per vertex in the approximation. This counts the two bytes each for the x, y, and z coordinates and four bytes each for the, on average, six neighbor pointers of each vertex.

For the RTIN, the storage required for each node in the binary tree is two bytes to hold the height of the midpoint of the hypotenuse and eight bytes for the two child pointers. This is a single surface approximation; the leaves of the tree represent triangles in the approximation. Thus, we do not need to store neighbor information in the nodes. To calculate the neighbor of a triangle, we calculate its same-size neighbor and take the closest leaf to that neighbor (the neighbor itself, its parent, or its child). The number of triangles in the approximation (i.e. leaves in the binary tree) is approximately twice the number of vertices in the approximation. The total number of nodes in the tree is approximately twice the number of leaves. Thus the RTIN uses approximately 40 bytes per vertex in the approximation.

For the gridded DEM the storage requirement is approximately two bytes per point in the approximation, since only the height values need to be stored.

The one instance in which the TIN approximation does not achieve the best performance of the three is when the RMS error of the approximation is considered as a function of the memory used. In this case, the gridded DEM achieves the best error for a given memory size (figure 12). This result supports Kumler's observations and, indeed, his error measures are similar to the RMS error measure. That the gridded DEM produces small error may be largely attributed to the fact that its triangles are, on average, quite small.

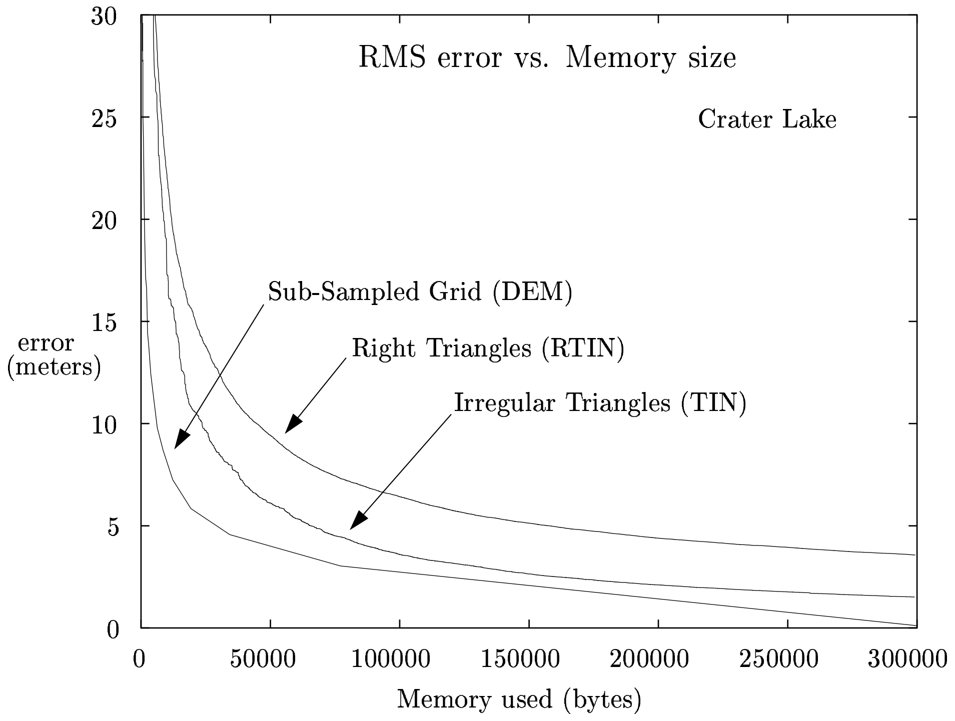

Figure 12: Root mean squared error versus memory usage. Crater Lake data set.

The performance of the RTIN as a single surface approximation does not match the performance of the general TIN approximation scheme. This will always be the case when performance is measured as a function of the number of points in the approximation. There is some hope, however, that other means of storage can bring down the memory requirements of the RTIN to the point where it competes with the TIN on single surface approximations when performance is a function of the memory used. For example, the labels of the triangles in the approximation may be hashed rather than stored in a binary tree. This avoids allocating space for internal nodes that are extraneous in a single surface approximation.

The data for the figures shown above are elevation data for Crater Lake West in southern Oregon obtained from Garland and Heckbert [24]. The results are similar to those obtained for other areas such as Yakima, Washington; Tucson, Arizona; and Reno, Nevada. We obtained these data sets and others from the U.S. Geological Survey [25]. We obtained the code that was used to produce the irregular TINs from Garland and Heckbert [24].

7.2 Hierarchical Performance

The rather poor performance of the RTIN structure in describing a single approximation is in contrast to its performance as a hierarchy for achieving multiple levels of detail in an interactive environment.

An interactive visualization system, called TopoVista, incorporating the ideas presented in this paper may be downloaded from the TopoVista web site [26]. All experimental results presented in this section are from this implementation.

The system on which we ran these experiments is an SGI Indigo2 with a High Impact graphics board and 128 megabytes of main memory, running IRIX 6.5. The program uses the OpenGL

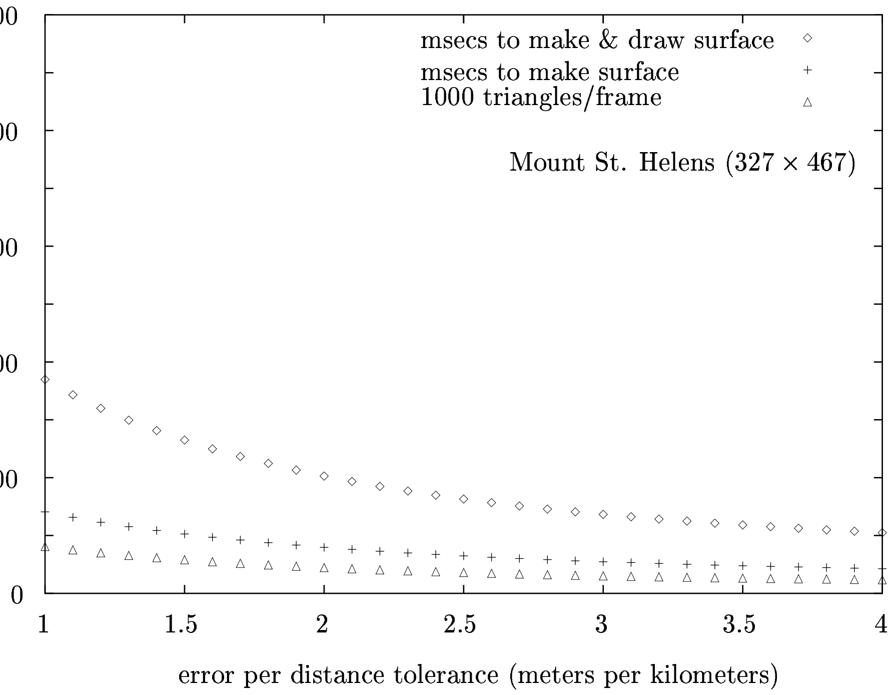

Figure 13: The average time to construct and time to construct and draw a surface approximation of Mount St. Helens as a function of the error per distance tolerance. The graph also shows the number of triangles (in 1,000's) in the approximation.

graphics package and the OpenGL Utility Toolkit, GLUT.

We present results for a small elevation model (figure 13), Mount St. Helens (327×467 data points), and a larger elevation model (figure 14), Mount Rainier East (977×1405 data points). In both cases, we measured the time to construct a surface approximation, the time to both construct and display the approximation, and the number of triangles in the approximation. We obtained these numbers by calculating the total time (and number of triangles drawn) to move the eye position around the perimeter of the elevation model, looking toward the center, and then dividing by the number of approximations calculated during this circuit. We repeated this experiment for varying error per distance tolerances. The tolerance specifies the vertical discrepancy, in meters, between the approximation and the original elevation model that is acceptable per one kilometer distance from the observer. Thus, an error per distance tolerance of 2.1 means that a triangle that is two kilometers from the observer, and within their field of view, must have a vertical discrepancy of at most 4.4 meters. In addition, in both cases, we insist that the approximation be at full detail within the observers field of view to a distance of 700 meters.

Allowing an error per distance tolerance of 1.0 within the observer's field of view resulted in approximately 7.5 (Mount St. Helens) or 25 (Mount Rainier) times fewer triangles than are in the triangulation of the original data points. With the tolerance at 4.0, the decrease was by a factor of 26 (Mount St. Helens) or 80 (Mount Rainier). Again these numbers are averages over all approximations from observer positions on the perimeter.

The following figures show the displayed RTIN approximations from the same eye position at two extreme error tolerances for Mount St. Helens before the May 18, 1980 eruption (figures 17 and 19) and after (figures 18 and 20), as well as the full detail versions of the same views.

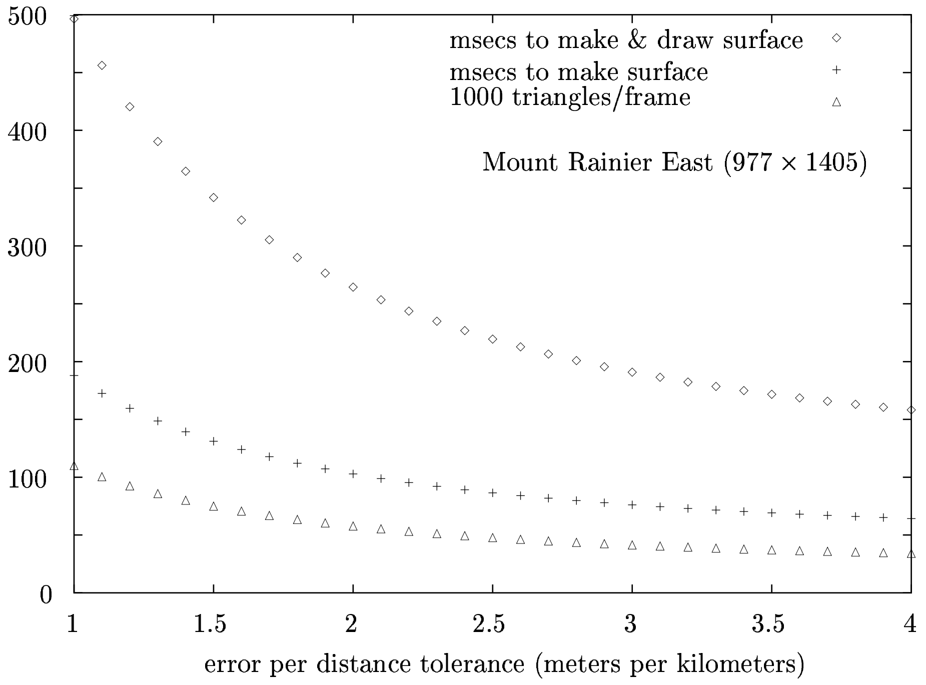

Figure 14: The average time to construct and time to construct and draw a surface approximation of Mount Rainier East as a function of the error per distance tolerance. The graph also shows the number of triangles (in 1,000's) in the approximation.

8 Comparison to Previous Work

Structures, similar to the RTIN structure, that support multiple approximations with multiple levels of detail have been discovered independently by Lindstrom, Koller, Ribarsky, Hodges, Faust, and Turner [6] and by Puppo [7, 8].

Lindstrom et al. [6] describe an interactive visualization system whose fundamental data structure is a hybrid between a RTIN and a sub-grid. The surface is partitioned into rectangular blocks each of which is regularly sub-sampled (like a sub-grid) then, within each block, a bottom-up reduction process eliminates vertices (data points) that are sufficiently well-approximated, forming a RTIN-like surface of right-angled isosceles triangles. Special care is taken at block boundaries to insure that no gaps are created. Many details of the data structure are left open, but space usage of between six and 28 bytes per data point are quoted, including two bytes for the input elevation. This compares favorably with our pure RTIN approach, that requires between ten and 14 bytes per data point. The difference in space usage between the two methods can be attributed to the fact that the RTIN hierarchy contains approximately four times as many triangles as data points. We store an error measure (and neighbor information) for each triangle while Lindstrom et al. store a single error measure for each data point. As a result, we can guarantee that our surface approximation is within a given tolerance (in our case, vertical discrepancy) of the original elevation model, while Lindstrom et al. cannot. It should be noted that both our ten byte version and Lindstrom et al.'s six byte version compress the surface error from its full detail two-byte representation to a one-byte representation, incurring a loss of precision.

Puppo [7, 8] describes an RTIN-like tree structure as a special case of a more general framework of multi-resolution models. He gives no experimental performance results for these structures, but

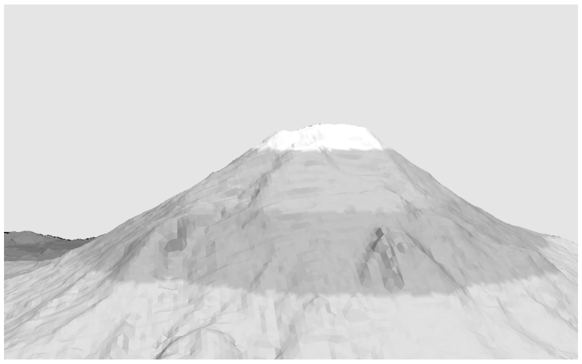

Figure 15: Full detail, pre-1980 Mount St. Helens (305,000 triangles).

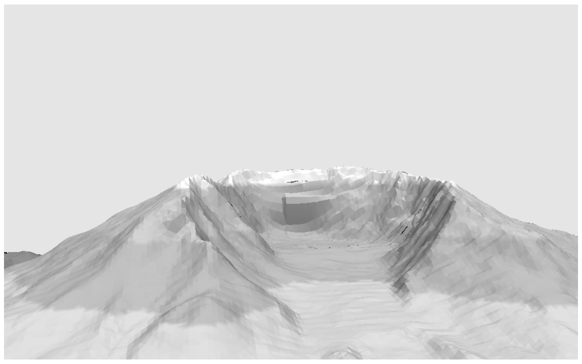

Figure 16: Full detail, post-1980 Mount St. Helens (305,000 triangles).

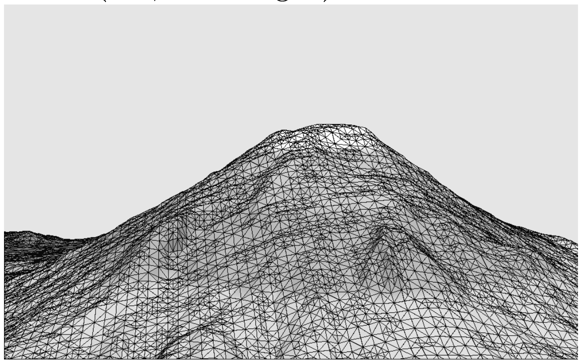

Figure 17: Tolerance = 1m per km (44,500 triangles).

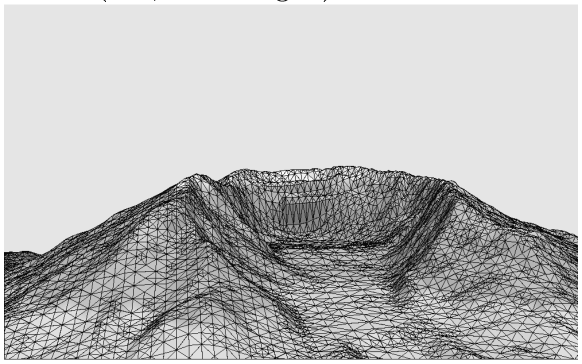

Figure 18: Tolerance = 1m per km (39,800 triangles).

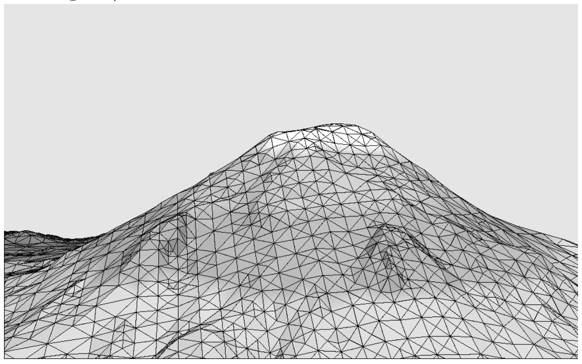

Figure 19: Tolerance = 4m per km (10,600 triangles).

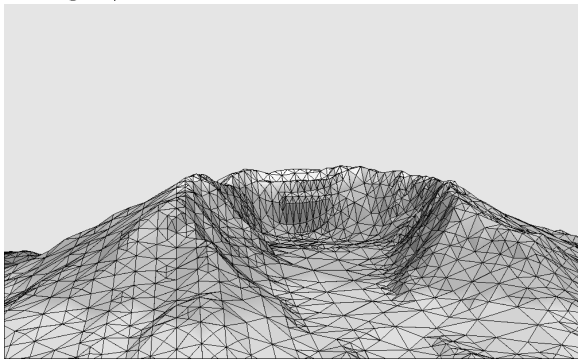

Figure 20: Tolerance = 4m per km (10,200 triangles).

does list, without detail, some of the operations, such as neighbor calculation, that such a structure should support.

9 Conclusions

In this paper, we present a type of surface approximation that is a restricted form of a triangulated irregular network; it consists solely of right-angled isosceles triangles, and is therefore called a right-triangulated irregular network or RTIN. We describe how a RTIN may be stored efficiently, and yet still support operations such as traversal from triangle to adjacent triangle that are required by most applications operating on surfaces.

The performance of RTINs for single surface approximation is close to that of TINs but, as one might expect, the accuracy achieved by a RTIN as a function of the number of points in the approximation is less than the accuracy of a TIN. Rather more surprising and disappointing is that the performance of a RTIN is worse than a TIN as a function of the memory used by the approximation. More efficient storage schemes for RTINs are possible and further research is needed to see if such RTIN representations can out-perform TINs.

RTINs perform much better as a representation of many approximations in a hierarchy that supports multiple level-of-detail approximations. The storage scheme we describe for a single surface approximation actually contains all coarser RTIN approximations. We describe a method for extracting an appropriate approximation quickly and illustrate its use in a system, TopoVista [26], that permits interactive visualization of USGS DEM data.

References

[1] University Libraries University of Washington State Geospatial Data Archive. Mount Rainier East. http://wagda.lib.washington.edu.

[2] L. De Floriani, P. Marazano, and E. Puppo. Multiresolution models for topographic surface description. *The Visual Computer*, 12(7):317–345, 1996.

[3] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996.

[4] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick P. Brooks, Jr., and William Wright. Simplification envelopes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 119–128. ACM SIGGRAPH, Addison Wesley, August 1996.

[5] E. Puppo and R. Scopigno. Simplification, LOD and multiresolution principles and applications. In *EUROGRAPHICS 97 Conference Proceedings*, volume 16. Blackwell Publishers, 1997.

[6] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. Technical report, Graphics, Visualization, & Usability (GVU) Center, Georgia Tech, 1996. ftp://ftp.gvu.gatech.edu/pub/gvu/tr/96-02.ps.Z.

[7] E. Puppo. Variable resolution triangulations. Technical Report 12/96, Institute for Applied Mathematics, National Research Council, Genova (Italy), November 1996. http://www.ima.ge.cnr.it/STAFF/PUPPO/PS/ima9612.ps.gz.

[8] E. Puppo. Bin-triangulations: Bridging the power of quadtrees and multiresolution triangulated models. manuscript, 1996.

[9] S. B. M. Bell, B. M. Diaz, F. Holroyd, and M. J. Jackson. Spatially referenced methods of processing raster and vector data. *Image and Vision Computing*, 1(4):211–220, 1983.

[10] A. V. Šubnikov. K voprosu o strenii kristallov. *Bulletin de l'Academie imperiale des sciences de St. Petersbourg, series VI*, 10:755–779, 1916.

[11] F. Laves. Ebenenteilung und koordinationszahl. *Zeitschrift für Kristallographie*, 5:68–105, 1931.

[12] D. J. Hebert and HyungJun Kim. Image encoding with triangulation wavelets. In *Proceedings of SPIE*, volume 2569(1), pages 381–392. The International Society for Optical Engineering, 1995. ftp://www.math.pitt.edu/pub/djh/articles/ietw/ietw-t.ps.

[13] D. J. Hebert. Interlaced quadtrees and binary triangulations. In Progress Lecture Notes, 1996. ftp://poincare.math.pitt.edu/pub/djh/articles/iqbt/000iqbt.ps.

[14] D. J. Hebert. Symbolic local refinement of tetrahedral grids. *Journal of Symbolic Computation*, 17(4):457–472, May 1994.

[15] L. Oliker, R. Biswas, and R. C. Strawn. Parallel implementation of an adaptive scheme for 3D unstructures grids on the SP2. Technical Report 96.11, Research Institute for Advanced Computer Science, NASA Ames Research Center, May 1996. To appear in *Proceedings of the 3rd International Workshop on Parallel Algorithms for Irregularly Structured Problems.*

[16] Hanan Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, 1989.

[17] B. Von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. In *SIGGRAPH 87 Conference Proceedings*. ACM SIGGRAPH, Addison Wesley, 1987.

[18] M. P. Kumler. *An Intensive Comparison of Triangulated Irregular Networks (TINs) and Digital Elevation Models (DEMs)*. University of Toronot Press, Inc., 1995. ISSN 0317-7173.

[19] M. Bertolotto, L. De Floriani, and P. Marzano. Pyramidal simplicial complexes. In *Proceedings 3rd ACM Symposium on Solid Modeling and Applications*, May 1995.

[20] M. de Berg and K. Dobrindt. On levels of detail in terrains. In *Proc. 11th Annual ACM Symp. on Computational Geometry*, June 1995. Also available as Utrecht University tech report UU-CS-1995-12, http://www.cs.ruu.nl/docs/research/publication/TechRep.html.

[21] J. W. J. Williams. Algorithm 232. *Communications of the ACM*, 7(6):347–348, June 1964.

[22] S. Yu, M. van Kreveld, and J. Snoeyink. Drainage queries in TINs: From local to global and back again. In *7th Symposium on Spatial Data Handling*, 1996. http://www.cs.ubc.ca/spider/snoeyink/papers/drainage.ps.gz.

[23] R. J. Fowler and J. J. Little. Automatic extraction of irregular network digital terrain models. *Computer Graphics (SIGGRAPH '79 Proceedings)*, 13(2):199–207, August 1979.

[24] M. Garland and P. S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical report, Carnegie Mellon University, 1995. Tech report and C++ code: http://www.cs.cmu.edu/ garland/scape.

[25] U.S. Geological Survey. *U.S. GeoData.* http://edcwww.cr.usgs.gov/doc/edchome/ndcdb/ndcdb.html.

[26] TopoVista. http://www.cs.arizona.edu/topovista/.