Right Triangular 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 allow the approximation to precisely represent the surface in certain areas while coarsely approximating the surface in others. 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.

There are many instances when an approximation to a surface is preferable to the true surface. Perhaps the most obvious is when storage space is limited. If this were the only concern then one solution might be to "approximate" the surface with a compressed representation that can be used to reconstruct the entire, full resolution surface when needed. More commonly, we need to simplify the representation not only to reduce storage space but also to make analysis faster and the results of analysis more comprehensible. Thus we need to be able to run analysis programs using the approximate surface; something which is not typically possible with generically compressed data. Furthermore, the approximation should be constructed so that the answers obtained in the analysis approximate the answers we would obtain for the true surface.

The structure we describe provides a framework that contains many approximations of varying levels of detail. One reason to construct approximation frameworks rather than single approximations is speed. It is often faster to extract a suitable approximation from an existing framework than to construct a suitable approximation from scratch. This is important when many different approximations are required in rapid succession.

The framework we propose in this paper is a hierarchy of approximations. Hierarchy implies that the framework contains a range of approximations, from coarse to fine. In addition, our hierarchy allows a mix of coarse and fine level representations within a single approximation. This allows

*Research partially supported by NSERC Canada International Fellowship.

us to obtain high levels of detail in certain regions of the surface without forcing us to sample 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.

We describe the hierarchy and illustrate its use 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. The goal is to avoid discontinuities or gaps in the surface at the transitions from coarser to finer approximations. This problem is very noticeable in interactive applications. It is common when linear interpolation is used over non-triangular regions.

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 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) and as a way to represent a single (non-hierarchical) surface approximation (appendix A).

We start by reviewing a portion of the large body of work that has been done on surface approximation. We then review two popular surface representations, and describe hierarchies based on these representations. This sets the stage for our discussion of the Right Triangular Irregular Network approximation (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 *multiresolution 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 [6]. This survey does not refer to a large body of very recent work that has appeared in SIGGRAPH96 [12, 3] and EuroGraphics96, both of which had sessions on multiresolution surface modeling. The most relevant work to our approach that has appeared in the surface approximation literature is the work by Lindstrom et al. [15] and Puppo [18, 17]. Both independently discovered hierarchies similar to the one discussed in this paper.

Another source of related work comes from the study of general lattice structures. Bell et al. [1] cite work on crystal lattice structures by Šubnikov in 1916 [21] and Laves [14] 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 [11, 10] and adaptive mesh generation for finite element solvers in two and higher dimensions [9, 16]. 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 [19].

3 DEM and TIN

Approximations of height data typically come in one of two types: gridded DEM or TIN. A gridded DEM (Digital Elevation Model) 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 (Triangular Irregular Network) are an arbitrary subset of the original data points. In order to specify this subset, the x, y coordinates 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 [13] argues that to achieve a certain degree of accuracy in the representation of a height field, DEMs 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 DEMs and TINs may shed some light on this issue (appendix A).

4 DEM and TIN hierarchies

Both DEM and TIN approximations may be organized into hierarchies. A hierarchy of DEM 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 [2, 4]. 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 can make interpolation less accurate. However, 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 retriangulate 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 at the price of storage. DeBerg and Dobrindt cite memory usage of 132 bytes per data point[1].

Our goal is to provide a hierarchy of surface approximations which is a compromise between the DEM and the TIN approximations. We hope that a more regular "grid-like" TIN will have the advantage of smaller space requirement and adaptability to terrain.

5 RTIN Hierarchy

Our hierarchical surface representation contains partitions of the two dimensional square into right-angled, isosceles triangles. We refer to this partition as a Right Triangular Irregular Network (RTIN). Such partitions have been studied in relation to crystal lattices [1], image encoding [11], and, in fact, geographic visualization systems [15, 18, 17]. 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. 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. In this case, linear interpolation over R is not well defined and 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

[1]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.

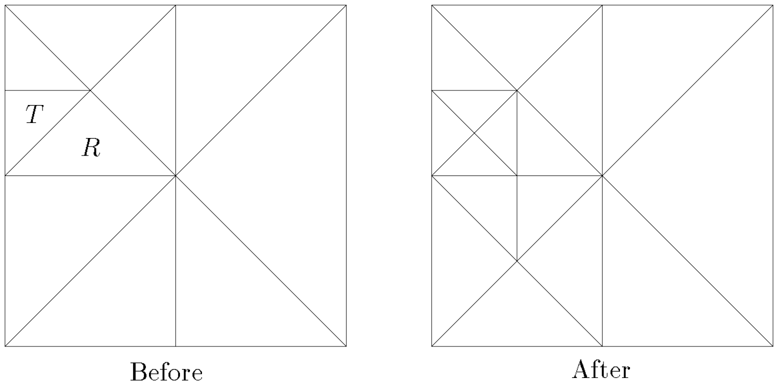

Figure 1: Splitting triangle T.

(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 [1] after F. Laves. The name lists the degree of the vertices of 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 pieces of the same shape.

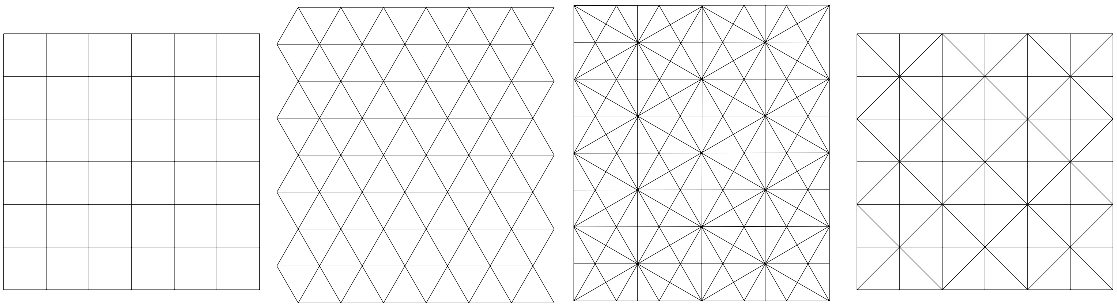

Figure 2: Pieces of Laves nets of type $[4^4]$, $[6^3]$, [4.6.12], and $[4.8^2]$.

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.

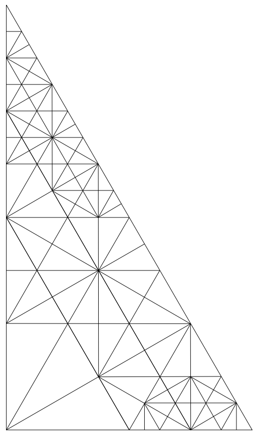

Figure 3: Non-uniform partition based on the [4.6.12] Laves net.

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 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, its vertices are equally spaced grid points, so 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.

5.1 Data Structure

The procedure by which the RTIN partitions were defined suggests the structure that is used to store them. Since each triangle in a partition can be divided into two similar pieces, an obvious choice for a storage structure is a binary tree. That is, each region in the partition has an associated node in a binary tree.

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 triangular regions in the square. The label of a region is a description of the path to its corresponding node in the binary tree. The path description is the

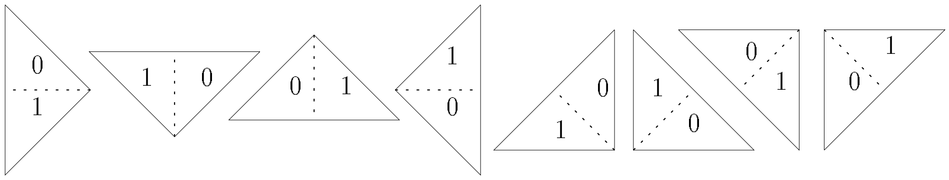

Figure 4: The possible triangle orientations and their left and right children. Left and right are denoted by 0 and 1 respectively.

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. See figure 5.

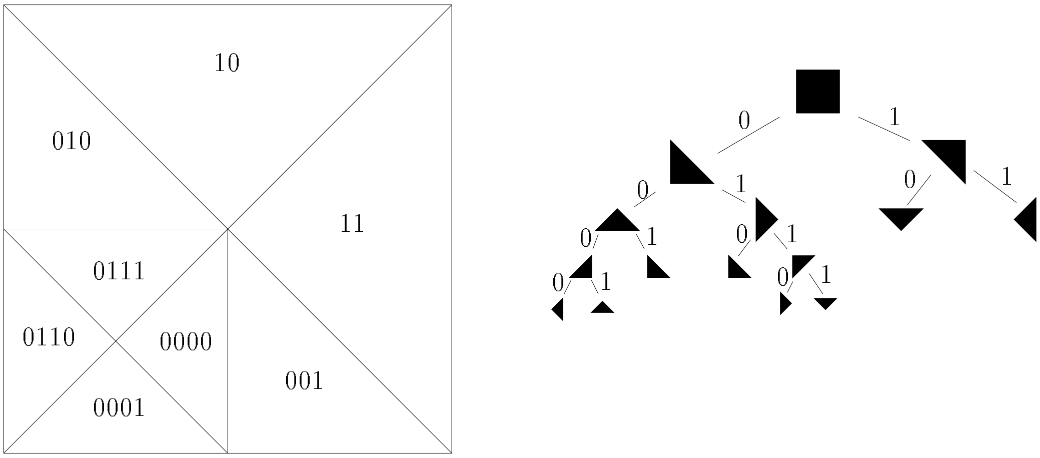

Figure 5: The binary tree representation of a triangulation in the hierarchy.

The advantage of this representation is that the x,y coordinates of the vertices of a triangle can be easily calculated from its label. The coordinates do not need to be explicitly stored as is the case with a general TIN. In this way, the RTIN resembles the regularly subsampled grid in which the x,y coordinates of a point can be calculated from the point's indices in the array of height values. However, unlike the regularly subsampled grid, the RTIN allows nonuniform sampling; some regions may be more densely subsampled 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 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 data. One possible location in which to store that height information is in the node corresponding to the parent triangle. 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.

A straightforward implementation of this binary tree structure is inefficient in its use of space.

Each node requires two pointers to its children as well as space for the height of the midpoint of its hypotenuse. This, along with the fact that there are twice as many nodes in the tree as triangles in the approximating surface, makes this method of surface approximation more space intensive than a TIN representation. See appendix A 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 is concerned with representing a hierarchy of surface approximations rather than just a single approximation. In some sense, the RTIN is tailor-made for hierarchical representations since, if it contains a surface approximation at a particular level of detail, it contains every coarser approximation.

We assume that the hierarchy should permit full level of detail in any region. Thus, the finest level of detail should be the original input data. If the original input was a $2^k+1 \times 2^k+1$ array, the binary tree representing the hierarchy is complete. 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, 0 and 00 both representing the same integer can easily be solved by prepending a 1 to the node label. The addressing scheme obtained in this way corresponds to the typical implicit array representation of a binary heap [22]. By using the array representation, we eliminate child pointer storage.

Since we require the hierarchy to contain the full level of detail, the height value of every x,y point must be available. Rather than store these heights in nodes of the binary tree, this data may be stored as a two dimensional array indexed by x and y coordinates (i.e. the original input representation), thus eliminating the storage needed in each node for the height of the midpoint of the corresponding triangle's hypotenuse.

At this point, one might ask what information is stored in the tree, since the pointer structure and the height values were the only previously required pieces of information.

The only vital piece of information that must be stored is an indication of the actual surface. 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[2] in order to determine how well the approximation fits the true surface.

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.

5.2 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 [23]. In order to accomplish this, the algorithm needs to determine

[2] 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

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*

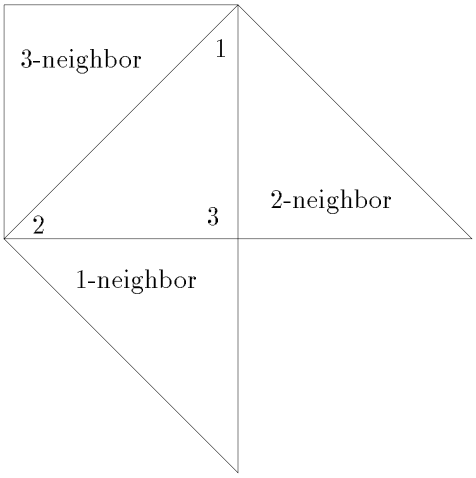

Figure 6: The i-neighbors of a triangle. In this case, all neighbors are the same size as the triangle.

of a triangle t is the triangle's i-neighbor in the uniform partition that contains t. The same-size neighbor may or may not be a part of the current 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 $i = 3$.

The following recursive procedures 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 label of the root. The symbol $\emptyset$ represents "no neighbor". The function $\circ$ is concatenation with the understanding that $\emptyset \circ 0 = \emptyset \circ 1 = \emptyset$.

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

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 appendix B).

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.

We assume that the rendering time is proportional to the number of triangles being drawn, rather than, for instance, their size. Such is the case for most graphics hardware/software packages. In order to achieve a certain level of performance (i.e. response time), the surface constructed in the approximation stage should have no more than a prescribed number of triangles (depending on this desired response time). Thus our algorithm should chose to refine the most poorly approximated parts of the surface until the allowed number of triangles is exceeded.

The alternative is to insist that the approximation everywhere satisfy 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 the previous case. A surface may require few triangles to achieve the desired overall fidelity in one case, while requiring many in another case. 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. We outline the method based on limiting the number of triangles. 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 heart of the algorithm is a loop which updates the surface representation in response to movements of the eye position. The algorithm constructs the approximation in a manner similar to the general greedy TIN construction method of Fowler and Little [7]. Their method incrementally adds the input point that is most poorly approximated to the approximation and then retriangulates the x,y projection of the points. 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. Thus, we order the triangles in the approximation by their "badness" and incrementally split the worst triangle (which may, in fact, cause several triangles to split). The measure of a triangle's badness may not simply be its error. A more appropriate measure for interactive visualization would also include some measure of the visibility of the triangle. For instance, triangles outside the current field of view, no matter how poorly they approximate the surface, are not seen by the observer and, hence, should have minimum badness.

The precise function used to calculate the badness of a triangle is not our main focus. One may choose an inverse distance weighting of the triangle's error, the area that the projection of the triangle occupies on the display screen, or other measures. (See [4] and [15] for examples.) One obvious common criteria is that triangles at the finest level of detail should not be refined.

As long as the number of triangles in the surface is below the allowed threshold, this procedure continues to split triangles in order of badness. When the threshold is reached, the triangles in the approximation are sent to the rendering software to be drawn.

7 Experimental results on hierarchical representations

An interactive visualization system incorporating the ideas presented in this paper may be downloaded from the first author's web site [5]. All experimental results presented in this session 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 6.2. The program uses the OpenGL graphics package and the OpenGL Utility Toolkit, GLUT.

We measured four aspects of the system:

1. The time to construct the RTIN hierarchy.

2. The memory required for the RTIN hierarchy.

3. The time to traverse the perimeter of the digital elevation model, looking toward its center, in single pixel steps. At each step the algorithm reconstructs the surface from the current eye position, redraws the 3-D perspective, and updates the field-of-view depiction in the 2-D relief view. If the 2-D relief view window has dimensions $x \times y$ then the algorithm renders $2(x + y)$ frames during the traversal.

4. The time to traverse the perimeter without redrawing the 3-D or 2-D views.

Time is wall-clock time in seconds. The following graph shows the performance of the system under two surface construction techniques.

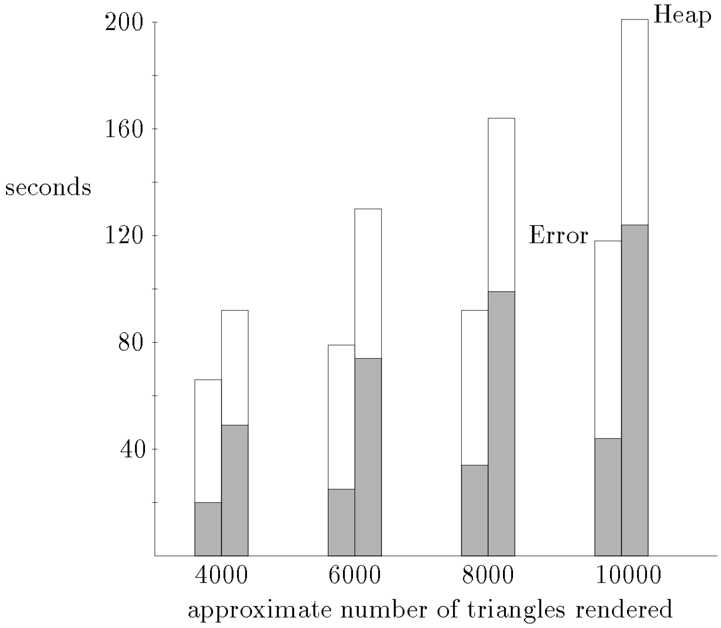

Figure 7: Time required to traverse DEM perimeter divided into surface approximation (gray) and rendering (white) for both refinement using a heap and refinement by scaled error measure. The total number of frames processed in each traversal was 55,296. The terrain data set contains 1025 $\times$ 1025 elevations. The hierarchy required 7 seconds to construct and consumed 17 Mbytes. The entire program used 23 Mbytes.

References

[1] 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.

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

[3] 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. held in New Orleans, Louisiana, 04-09 August 1996.

[4] 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.

[5] W. Evans. http://cs.arizona.edu/people/will/will.html.

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[13] 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.

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

[15] 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.

[16] 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.*

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

[18] E. Puppo. Variable resolution triangulations. Technical report, Institute for Applied Mathematics, National Research Council, Genova (Italy), 1996.

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

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

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

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

[23] 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.

A Single surface representations

In 1979, Fowler and Little proposed a scheme to select the vertices of a TIN [7]. 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 retriangulated (using a Delaunay triangulation) to form the next approximation.

We can, in spirit, mimic this greedy approach using right triangular partitions. Again the procedure is iterative, but at each iteration the triangle which contains the least well approximated point is split (perhaps causing other triangles to split) to obtain the next approximation.

This 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 RTINs, this greedy heuristic is rather simple to adopt: at each iteration the triangle with the maximum RMS error is chosen to be split. For TINs, finding the point that most decreases the RMS error is prohibitively expensive.

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 subsampled DEM is the worst performer under this measure (figure 8). 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.

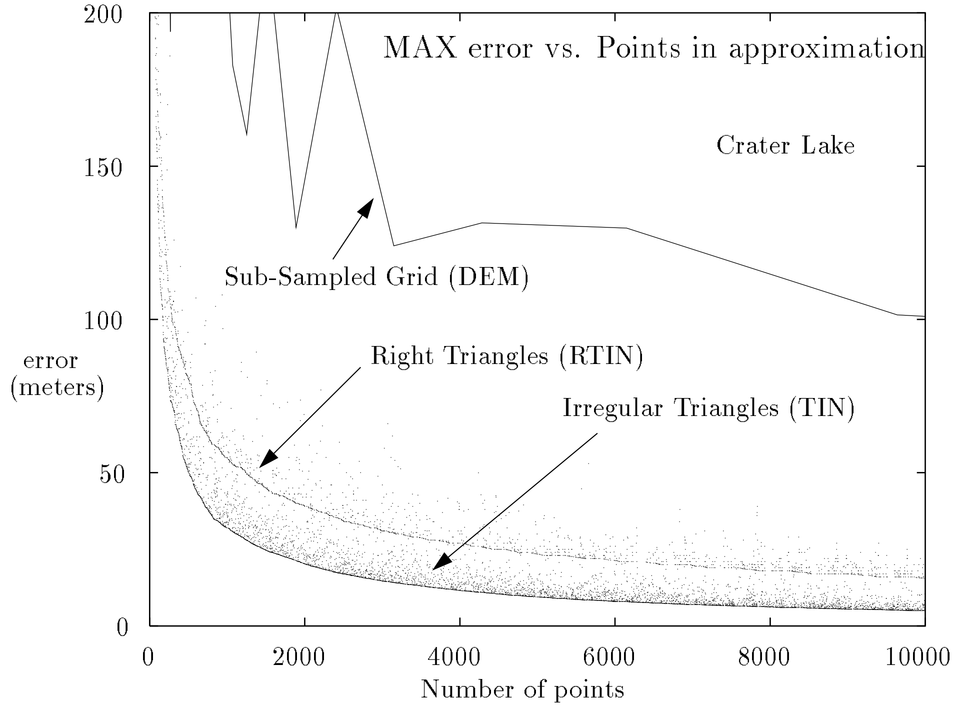

Figure 8: Maximum error versus number of points in approximation. Crater Lake data set.

We obtain a similar result when we consider the RMS (root mean squared) 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.

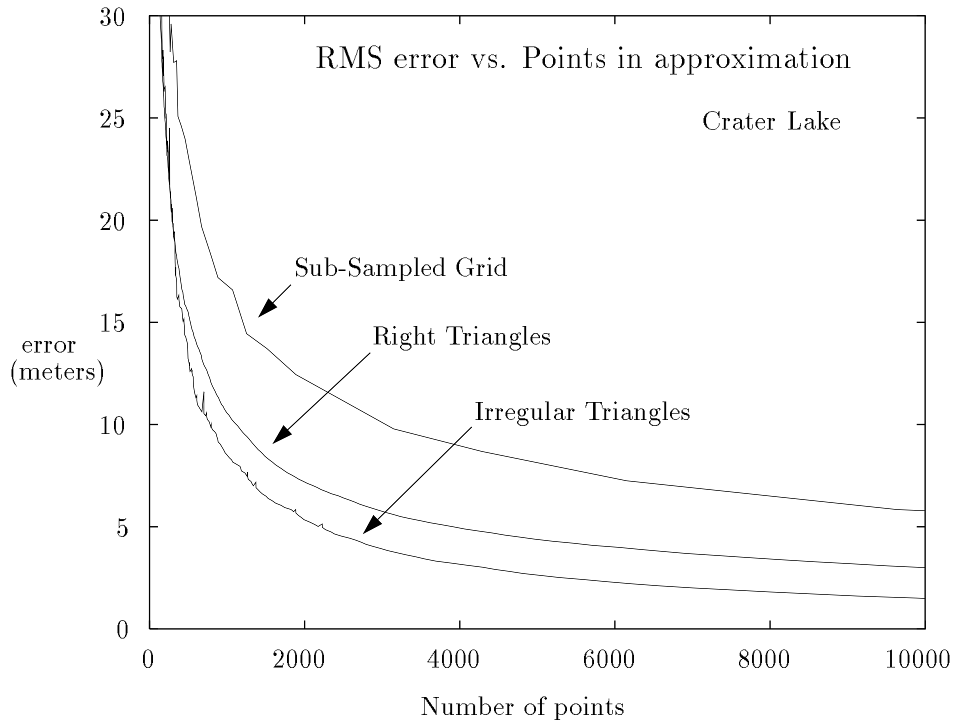

Figure 9: Root mean squared error versus number of points. Crater Lake data set.

Even when we consider the maximum error as a function of the amount of memory required to store the approximations, the irregular triangulation or TIN still outperforms both the RTIN approximation and the gridded DEM approximation. 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. This result

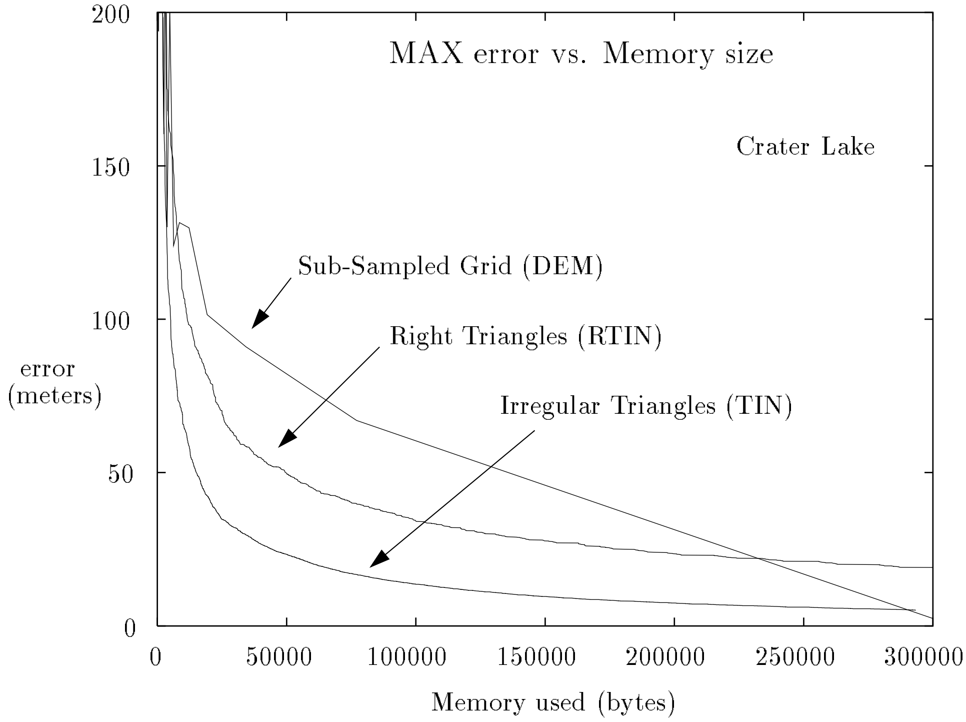

Figure 10: Maximum error versus memory usage. Crater Lake data set.

supports Kumler's observations and, indeed, his error measures are of a similar flavor 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.

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 is elevation data for Crater Lake in southern Oregon obtained from Garland and Heckbert [8]. The results are similar to those obtained for other areas such as Yakima, Washington, Tucson, Arizona, and Reno, Nevada. These data sets and others were obtained from the U.S. Geological Survey [20]. The code which was used to produce the irregular triangulations was obtained from [8].

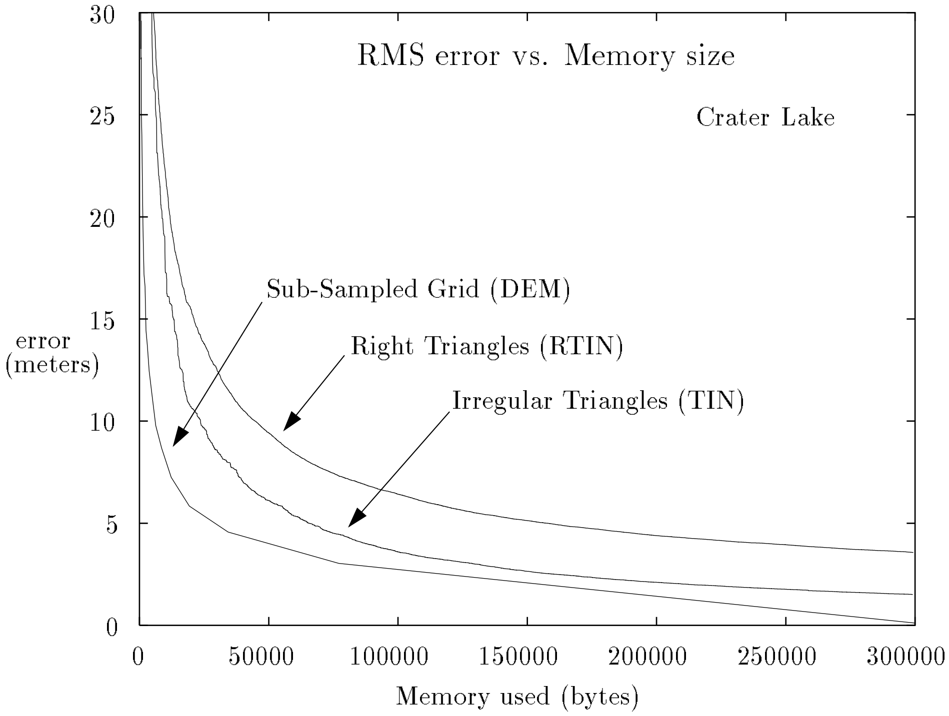

Figure 11: Root mean squared error versus memory usage. Crater Lake data set.

B Fast Neighbor Calculation Code

```
int neighbor[3][3] = {{0, 0, 0}, {0, 0, 0}, {0, 2, 1}};

#define SIEVE0 0xAAAAAAAAAAAAAAAA
#define SIEVE1 0x5555555555555555

/*  sameSizeNbr(t, i) -- return the label of the same size i-neighbor of t. */

int sameSizeNbr(int t, int i)
{
    register unsigned int a;
    register unsigned int b;
    register unsigned int c;
    register unsigned int p;
    register unsigned int n;

    if (t <= 2) {
        return neighbor[i-1][t];
    }
    p = t + 1;
    if (i == 1) {
        p = (p << 1) | 1;
    } else if (i == 2) {
        p = (p << 1);
    }
    c = ((p << 1) ^ p) & SIEVE0;
    a = c | (c << 1) | 1;
    b = (a + 1) ^ a;        /* most of the work is done by this addition */
    if (b < (p>>1)) {
        n = p ^ b;
    } else if (((p & SIEVE0) << 1) > p) {
        if (((p & SIEVE1) << 2) > p) {
            n = p ^ (b>>1);
        } else {
            n = p ^ (b>>3);
        }
    } else {
        return 0;
    }
    if (i != 3) {
        return (n >> 1) - 1;
    } else {
        return n - 1;
    }
}
```