Real-Time, Continuous Level of Detail Rendering of Height Fields

Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges
College of Computing and Graphics, Visualization, and Usability Center
Georgia Institute of Technology

Nick Faust
Center for GIS and Spatial Analysis Technologies and Georgia Tech Research Institute
Georgia Institute of Technology

Gregory Turner
Information Processing Branch
Army Research Laboratory

Abstract

We present an algorithm for real-time level of detail reduction and display of high-complexity polygonal surface data. The algorithm uses a compact and efficient regular grid representation, and employs a variable screen-space threshold to bound the maximum error of the projected image. The appropriate level of detail is computed and generated dynamically in real-time, allowing for smooth changes of resolution across areas of the surface. The algorithm has been implemented for approximating and rendering digital terrain models and other height fields, and consistently performs at interactive frame rates with high image quality. Typically, the number of rendered polygons per frame can be reduced by two orders of magnitude while maintaining image quality such that less than 5% of the resulting pixels differ from a full resolution image.

1 Introduction

Modern graphics workstations allow the display of thousands of shaded or textured polygons at interactive rates. However, many applications contain graphical models with geometric complexity still greatly exceeding the capabilities of typical graphics hardware. This problem is particularly prevalent in applications dealing with large polygonal surface models, such as digital terrain modeling and visual simulation systems.

In order to accommodate such complex surface models while still maintaining real-time display rates, methods for approximating the polygonal surfaces and using multiresolution models have been proposed [13]. Approximation algorithms can be used to generate multiple surface models at varying levels of detail, and techniques are employed by the display system to select the appropriate level of detail and render it appropriately.

In this paper we present a new level of detail display algorithm that is applicable to surfaces that are represented as uniformly-gridded height fields. The algorithm greatly reduces the number of polygons to be rendered, providing for real-time rendering of complex surfaces while meeting user-controlled constraints on image quality, and does not suffer from the limitations of previous algorithms.

- Our algorithm is fast. In typical visual simulation applications containing large, high-resolution terrain surface models, our algorithm allows sustained frame rates on the order of 20 frames per second.
- Our algorithm meets a user-specified image quality metric. The algorithm is easily controlled to meet an image accuracy level within a specified number of pixels. This parameterization allows for easy variation of the balance between rendering time and rendered image quality.
- Our algorithm provides for smooth, continuous changes between different surface levels of detail. The appropriate level of detail for an area of the surface is computed dynamically, in real-time, with no need

for expensive generation of multiple level of detail models ahead of time.

Related approaches to polygonal surface approximation and multiresolution rendering are discussed in the next section. The following sections of the paper describe the data structures and procedures for implementing the real-time continuous rendering algorithm. We conclude the paper by empirically evaluating the algorithm with results from its use in a typical application.

2 Related Work

A large number of researchers have developed algorithms for approximating terrains and other height fields using polygonal meshes. Heckbert and Garland [14] review many of these surface simplification methods, categorizing the algorithms into several groups, including uniform grid methods (which use a regular grid of height samples), hierarchical subdivision methods, and general triangulation methods (such as those that employ Delaunay triangulation techniques).

Much of the previous work on polygonalization of terrain-like surfaces has concentrated on triangulated irregular networks (TINs). A number of different approaches have been developed to create TINs from height fields using Delaunay and other triangulations [8, 3, 17, 19, 18, 10]. TINs allow variable spacing between vertices of the triangular mesh, approximating a surface at any desired level of accuracy with fewer polygons than other representations. Fowler and Little [8] construct TINs characterized by certain "surface specific" points and critical lines, allowing the TIN representation to closely match important terrain features. However, the algorithms required to create TIN models are generally computationally expensive, prohibiting use of dynamically created TINs at interactive rates.

Regular grid surface polygonalizations have also been implemented as terrain and general surface approximations [2, 6]. Such a uniform polygonalization generally produces many more polygons than a TIN for a given level of approximation, but the grid representation is typically more compact. Regular grid representations also have the advantage of allowing for easier construction of a multiple level of detail hierarchy. Simply subsampling grid elevation values produces a coarser level of detail model, whereas TIN models generally require complete retriangulation in order to generate multiple levels of detail.

Other surface approximation representations include hybrids of these techniques, and Douglas's "richline" model. Douglas [4] located specific terrain features such as ridges and channels in a terrain surface data set, representing the surface with line segments from these "information rich" lines. This method generates only a single surface approximation, however, and is not easily adapted to produce multiresolution models. Gross et al. [12] use a wavelet transform to produce adaptive surface meshing from uniform grid data, allowing for local control of the surface level of detail.

The issue of "continuous" level of detail representations for models has been addressed both for surfaces and more general modeling. Taylor and Barret [22] give an algorithm for surface polygonalization at multiple levels of detail, and use "TIN morphing" to provide for visually continuous change from one resolution to another. Many visual simulation systems handle transitions between multiple levels of detail by alpha blending two models during the transition period. Ferguson [7] claims that such blending techniques between adjacent levels of detail may be visually distracting, and discusses a method of Delaunay triangulation and triangle subdivision which smoothly matches edges across areas of different level of detail.

3 Motivation

The algorithm presented in this paper is a powerful hybrid of algorithms that combines the large complexity reduction obtained in TIN constructions, and the flexibility provided by regular grids, while avoiding many of the drawbacks inherent in these types of algorithms. By extending the regular grid representation to allow polygons to be recursively "fused" where appropriate, a larger polygon reduction can be obtained. The notion of a continuous level of detail obtained via small, incremental changes to the mesh polygonalization, here plays an important role, and provides a rigid framework for accommodating frame rate consistency algorithms. The simplification algorithm guarantees to meet a user-specified image accuracy constraint, which can be modified interactively to obtain desirable visual results and/or frame update rates.

Desirable characteristics for a real-time, level of detail (LOD) algorithm for height fields include:

(i) The mesh geometry and the components that describe it should at any instant be directly and efficiently queryable. Polygonal surfaces should be traceable and allow for fast spatial indexing.

(ii) Dynamic changes to the geometry of the mesh should not significantly impact the performance of the system. That is, recomputation of parameters and/or

repolygonalization or reconstruction of the geometry should be virtually instantaneous.

(iii) High frequency data such as localized convexities and concavities, and/or local changes to the geometry, should remain local without having a larger global effect on the complexity of the model.

(iv) Small changes to the view parameters (e.g. viewpoint, view direction, field of view) should lead only to small changes in complexity in order to minimize uncertainties in prediction and allow maintenance of (near) constant frame rates.

(v) The algorithm should provide a means of bounding the loss in image quality incurred by the approximated geometry of the mesh. That is, there should exist a consistent and simple relationship between the input parameters to the LOD algorithm, and the resulting image quality.

Note that some applications do not require the satisfaction of all of these criteria. However, these are all general issues and, in particular, need to be addressed in the design of a level of detail algorithm for polygonal terrain meshes, which is our target application domain.

Most contemporary approaches to level of detail management fail to meet at least one of these five criteria. TIN models, for example, do not in general meet the first two criteria. Generation of even modest size TINs require extensive computational effort, and to our knowledge, cannot be done in real-time. Because TINs are non-uniform in nature, surface following (e.g. for animation of objects on the surface) and intersection (e.g. for collision detection, selection, and queries) are hard to handle efficiently due to the lack of a spatial organization of the mesh polygons. The importance of (ii) is relevant in many applications, such as games and military applications where explosions dynamically deform the terrain.

The most common drawback of regular grid representations is that the polygonalization is seldom optimal, or even near optimal. Large, flat surfaces may require the same polygon density as small, rough areas do. This is due to the sensitivity to localized, high frequency data within large, uniform resolution areas of lower complexity. (Most level of detail algorithms require that the mesh is subdivided into rectangular blocks of polygons to allow for fast view culling and level of detail selection, and the blocks may also serve as a unit of transfer in data paging.) Hence, (iii) is violated as a small bump in the mesh may force higher resolution data than is needed to describe the remaining area of a block. This problem may be alleviated by reducing the overall complexity and applying temporal blending, or morphing, between different levels of detail to avoid "popping" in the mesh [15, 22].

Common to typical TIN and regular grid LOD algorithms is the discreteness of the levels of detail. Often, only a relatively small number of models for a given area are defined, and the difference in the number of polygons in successive levels of detail may be quite large. When switching between two levels of detail, the net change in the number of rendered polygons may amount to a rather large fraction of the given rendering capacity, and may cause rapid fluctuations in the frame rate, making frame rate consistency and prediction problematic.

Many LOD algorithms fail to recognize the need for an error bound in the rendered image. While many simplification methods are mathematically viable, the level of detail generation and selection are often not directly coupled with the screen-space error resulting from the simplification. Rather, these algorithms are used to *characterize* the data with a small set of parameters that, in conjunction with viewpoint distance and view angle, may be used to select what could be deemed appropriate levels of detail. Examples of such algorithms include TIN simplification [8], feature (e.g. peaks, ridges, and valleys) identification and preservation [21, 4], and frequency analysis/transforms such as wavelet simplification [5, 12]. These algorithms do, in general, not provide enough information to derive a tight bound on the maximum error in the projected image. For example, given a set of TINs for an area, it is not immediately obvious which model should be rendered for a given viewpoint unless the factors that contribute to the error in the image can be parameterized, stored, and later evaluated in screen-space coordinates. If image quality is important and "popping" effects need to be minimized in animations, the level of detail selection should be based on a user-specified error tolerance measured in screen-space, and should preferably be done on a per polygon/vertex basis.

The algorithm presented in this paper satisfies all of the above criteria. Some key features of the algorithm are: flexibility and efficiency– the internal representation is a regular grid; localized polygon densities– the resolution within a block may vary; screen-space error-driven LOD selection–a single threshold determines the image quality; and continuous level of detail, which will be discussed in the following section.

3.1 Continuous Level of Detail

Continuous level of detail has recently been used to describe a variety of properties [7, 17, 22], some of which will be described below. As mentioned in (iii) and (iv) above,

it is important that the complexity changes smoothly between consecutive frames, and that the simplified geometry doesn't lead to gaps or popping in the mesh. In a more precise description of the term *continuity* in the context of level of detail, the continuous function, its domain, and its range must be clearly defined. This function may be one of the following:

(i) The elevation function $z(x, y, t)$, where $x, y, t \in \mathbf{R}$. The parameter t may denote time, distance, or some other scalar quantity. This function is used to morph (blend) the geometries of two discrete levels of detail defined on the same area, resulting in a virtually continuous change in level of detail over time, or distance from the viewpoint to the mesh [22].

(ii) The elevation function $z(x, y)$ with domain $\mathbf{R}^2$. The function z is defined piecewise on a per block basis. When discrete levels of detail are used to represent the mesh, two adjacent blocks of different levels of detail may not align properly, and gaps along the boundaries of the blocks may be seen. The elevation z on these borders will not be continuous unless precautions are taken to ensure that such gaps are smoothed out.

(iii) The number of polygons rendered (after clipping) $n(\mathbf{v})$, where $\mathbf{v}$ is the viewpoint vector.[1] Since the image of n is discrete, continuity is here somewhat informally defined in terms of the modulus of continuity $\omega(\delta)$. We say that n is continuous iff $\omega(\delta) \rightarrow \varepsilon,\ \varepsilon \leq 1$ as $\delta \rightarrow 0$. That is, for sufficiently small changes in the viewpoint, the change in the number of polygons rendered is at most one. Note that this type of continuity does not explicitly imply that the polygon distribution over $\mathbf{R}^2$ changes smoothly with $\mathbf{v}$, e.g. if modeled as a knapsack problem where the number of polygons rendered is (near) constant, the "polygon density" may vary rapidly over a given area [9].

(iv) The "polygon density" or polygon distribution function $n(\mathbf{v}, A)$, where A is any fixed subset of $\mathbf{R}^2$. For a given area A, the number of polygons used to describe that area is continuous with respect to the viewpoint $\mathbf{v}$. Note that A does not necessarily have to be a connected set. There is a subtle difference between this type of continuity and (iii), in which A depends on $\mathbf{v}$.

[1] This vector may be generalized to describe other view dependent parameters, such as view direction and field of view.

Note that a continuous level of detail algorithm may possess one or more of these independent properties (e.g. (i) does in general not imply (iii), and vice versa). The algorithm presented here primarily fits the last two definitions of continuity, but has been designed to be easily extensible to cover the other two definitions.

4 Simplification

The simplification process used in this algorithm is reductive, meaning that many smaller polygons are successively removed and replaced with fewer, larger polygons. The polygons are here triangles formed by connecting vertices laid out on the rectilinear grid of integer-valued elevation points that constitutes the height field database. Conceptually, at the beginning of each rendered frame, the entire database at its highest resolution is considered. Wherever certain conditions are met, a pair of triangles is reduced to one single triangle, and the resulting triangle and its *co-triangle* (if one exists) are considered for further simplification in a recursive manner. In order to perform this simplification, the height field must first be triangulated, and triangle/co-triangle pairs must be identified. Figure 1 shows the height field triangulation for meshes of different dimensions, as well as the triangle/co-triangle pairs, where each pair is assigned a unique letter, e.g. $\triangle_{a_l a_r} = (a_l, a_r)$ is a pair. Second level pairs are defined as pairs of pairs of triangles, e.g. $\triangle_{a_l a_r}$ and $\triangle_{b_l b_r}$ form another triangle pair, denoted by $((a_l, a_r), (b_l, b_r))$. The smallest mesh representable using this symmetrical triangulation (the *primitive mesh*) has dimensions 3×3 vertices, and successively larger meshes are formed by grouping four smaller meshes in a 2×2 array configuration. This recursive definition of the mesh imposes two major constraints on the dimensions x_{dim} and y_{dim} of the *blocks* that make up the mesh: $x_{dim} = y_{dim} = 2^n + 1$ for some integer $n \geq 1$, although the resolutions (vertex spacings) x_{res} and y_{res} do not necessarily have to be the same. Depending on the implementation, these blocks could, for instance, represent the *quadnodes* in a *quadtree* [16]. All blocks have the same vertex dimensions (i.e. they contain the same number of vertices), but may differ in resolution and area. Adjacent blocks share the vertices that lie on the block edges.

The conditions under which a triangle pair can be coalesced into a single triangle are primarily described by the amount of change in slope between the two triangles. For triangles $\triangle ABE$ and $\triangle BCE$, $\angle ABE > \angle AEB$, and $\angle CBE > \angle CEB$, this change is measured by the vertical (z axis) distance $\delta_B = 2|B \Leftrightarrow \frac{A+C}{2}| = |2B \Leftrightarrow (A + C)|$, i.e.

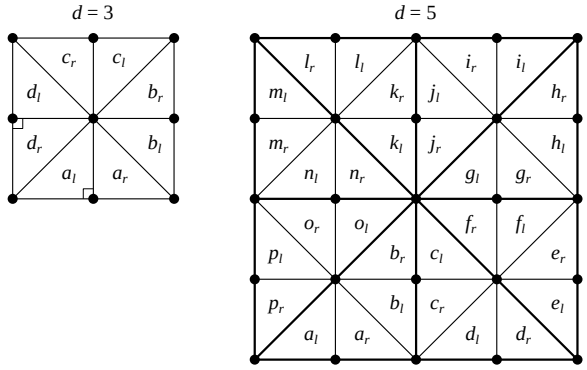

Figure 1: Triangulation of uniform height fields of dimensions $d = 3$ and $d = 5$.

twice the maximum vertical distance between $\triangle ACE$ and the triangles $\triangle ABE$ and $\triangle BCE$ (see Figure 2).[2] This is referred to as the vertex's *delta value*. In general, as the delta value increases, the chance of triangle fusion decreases. By projecting the *delta segment*, defined by B and the midpoint of $\overline{AC}$, onto the projection plane, one can determine the maximum perceived geometric (linear) error between the coalesced triangle and its "children." (Note that the error (change) in volume is proportional to the delta value; $\Delta V = \pm\frac{x_{res}y_{res}}{6}\delta$.) If this linear error is smaller than a given threshold, τ, the triangles may be fused. If the resulting triangle has a co-triangle (that is, if the co-triangle's error also is smaller than the threshold), this pair is considered for further simplification. This process is applied recursively until no further simplification can be made.

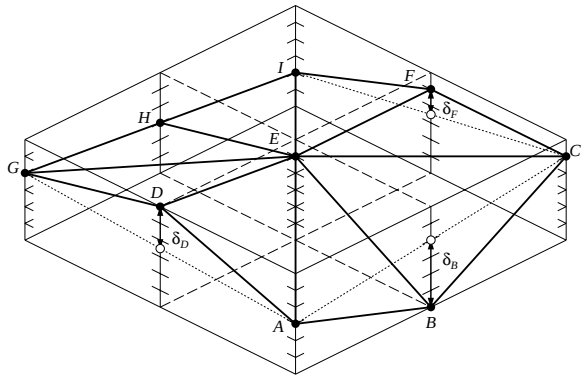

Figure 2: Geometric representation of delta values. $\delta_B = 8$, $\delta_D = 5$, $\delta_F = 3$, $\delta_H = 0$.

We now derive a formula for the length of the projected delta segment. Let $\mathbf{v}$ be the midpoint[3] of the delta segment, and define $\mathbf{v}^+ = \mathbf{v} + \left[\begin{array}{ccc} 0 & 0 & \frac{\delta}{4} \end{array}\right]$, $\mathbf{v}^- = \mathbf{v} \Leftrightarrow \left[\begin{array}{ccc} 0 & 0 & \frac{\delta}{4} \end{array}\right]$. Let $\mathbf{e}$ be the viewpoint and $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, $\hat{\mathbf{z}}$ be the orthonormal eye coordinate axes expressed in world coordinates. Furthermore, let n be the distance from $\mathbf{e}$ to the projection plane, and define λ to be the number of pixels per world coordinate unit in the screen x-y coordinate system. (We assume, for simplicity, that the pixel aspect ratio is 1:1.) The subscripts *eye* and *screen* are used to denote vectors represented in *eye coordinates* (after the view transformation) and *screen coordinates* (after the perspective projection) respectively. Then, the viewing matrix is defined as:

$$\mathbf{M} = \left[\begin{array}{cccc} \hat{x}_x & \hat{y}_x & \hat{z}_x & 0 \\ \hat{x}_y & \hat{y}_y & \hat{z}_y & 0 \\ \hat{x}_z & \hat{y}_z & \hat{z}_z & 0 \\ \Leftrightarrow\mathbf{e}\cdot\hat{\mathbf{x}} & \Leftrightarrow\mathbf{e}\cdot\hat{\mathbf{y}} & \Leftrightarrow\mathbf{e}\cdot\hat{\mathbf{z}} & 1 \end{array}\right].$$

Using these definitions, the following approximations are made:

- When projecting the vectors $\mathbf{v}^+$ and $\mathbf{v}^-$, their midpoint $\mathbf{v}$ is always assumed to be in the center of view, i.e. along $\Leftrightarrow\hat{\mathbf{z}}$. This approximation is reasonable as long as the field of view is relatively small. For a 60° field of view, the maximum error due to this approximation is less than 15%, and its effect is that projected objects away from the center of view appear relatively smaller than objects closer to the center of view. Hence, the projected delta segments that represent the errors in the triangle simplification become relatively smaller at the periphery of the screen, leading to relatively higher detail in the center of view. This artifact is often acceptable as the focus is often placed on the center of the screen, while the visual perception is degraded toward the periphery.
- In the perspective division $\frac{1}{-v_{eye_z}}$, we assume $v^+_{eye_z} \simeq v^-_{eye_z} \simeq v_{eye_z}$. This is a fair assumption since, in general, $\delta \ll ||\mathbf{e} \Leftrightarrow \mathbf{v}|| = \Leftrightarrow v_{eye_z}$.

According to the first approximation, the viewing matrix should be defined as:

[2] The reason for doubling this distance is due to its representation as an integer value.

[3] Rather than having to compute the midpoint of the delta segment at (x, y), one may safely substitute the vertex located at (x, y) for $\mathbf{v}$.

$$\mathbf{M} = \begin{bmatrix} \hat{x}_x & \hat{y}_x & \frac{e_x - v_x}{||\mathbf{e}-\mathbf{v}||} & 0 \\ \hat{x}_y & \hat{y}_y & \frac{e_y - v_y}{||\mathbf{e}-\mathbf{v}||} & 0 \\ \hat{x}_z & \hat{y}_z & \frac{e_z - v_z}{||\mathbf{e}-\mathbf{v}||} & 0 \\ \Leftrightarrow\mathbf{e}\cdot\hat{\mathbf{x}} & \Leftrightarrow\mathbf{e}\cdot\hat{\mathbf{y}} & \Leftrightarrow\mathbf{e}\cdot\frac{\mathbf{e}-\mathbf{v}}{||\mathbf{e}-\mathbf{v}||} & 1 \end{bmatrix}.$$

The length of the projected delta segment is then described by the following set of equations:

$$\begin{aligned} \mathbf{v}^+_{eye} \Leftrightarrow \mathbf{v}^-_{eye} &= \mathbf{v}^+\mathbf{M} \Leftrightarrow \mathbf{v}^-\mathbf{M} \\ &= (\mathbf{v}^+ \Leftrightarrow \mathbf{v}^-)\mathbf{M} \\ &= \begin{bmatrix} 0 & 0 & \frac{\delta}{2} & 0 \end{bmatrix}\mathbf{M} \\ &= \frac{\delta}{2}\begin{bmatrix} \hat{x}_z & \hat{y}_z & \frac{e_z - v_z}{||\mathbf{e}-\mathbf{v}||} & 0 \end{bmatrix} \\ \hat{x}_z^2 + \hat{y}_z^2 &= 1 \Leftrightarrow \left(\frac{e_z \Leftrightarrow v_z}{||\mathbf{e}\Leftrightarrow\mathbf{v}||}\right)^2 \end{aligned}$$

$$\begin{aligned} \delta_{screen} &= ||\mathbf{v}^+_{screen} \Leftrightarrow \mathbf{v}^-_{screen}|| \\ &= \frac{n\lambda\sqrt{(v^+_{eye_x} \Leftrightarrow v^-_{eye_x})^2 + (v^+_{eye_y} \Leftrightarrow v^-_{eye_y})^2}}{\Leftrightarrow v_{eye_z}} \\ &= \frac{n\lambda\sqrt{\left(\frac{\delta\hat{x}_z}{2}\right)^2 + \left(\frac{\delta\hat{y}_z}{2}\right)^2}}{||\mathbf{e}\Leftrightarrow\mathbf{v}||} \\ &= \frac{n\lambda\delta\sqrt{1 \Leftrightarrow \left(\frac{e_z - v_z}{||\mathbf{e}-\mathbf{v}||}\right)^2}}{2||\mathbf{e}\Leftrightarrow\mathbf{v}||} \\ &= \frac{n\lambda\delta\sqrt{\frac{(e_x-v_x)^2+(e_y-v_y)^2}{(e_x-v_x)^2+(e_y-v_y)^2+(e_z-v_z)^2}}}{2\sqrt{(e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2 + (e_z \Leftrightarrow v_z)^2}} \\ &= \frac{k\delta\sqrt{(e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2}}{(e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2 + (e_z \Leftrightarrow v_z)^2} \quad (1) \end{aligned}$$

$$\delta^2_{screen} = \frac{k^2\delta^2((e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2)}{((e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2 + (e_z \Leftrightarrow v_z)^2)^2} \quad (2)$$

For performance reasons, δ^2_{screen} is compared to τ^2 so that the square root can be avoided. The inequality that defines the simplification condition can be reduced to a few additions and multiplications:

$$\begin{aligned} &\frac{k^2\delta^2((e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2)}{((e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2 + (e_z \Leftrightarrow v_z)^2)^2} \leq \tau^2 \Leftrightarrow \\ &\delta^2((e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2) \leq \\ &\quad \kappa((e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2 + (e_z \Leftrightarrow v_z)^2)^2 \quad (3) \end{aligned}$$

where $\kappa = \frac{\tau^2}{k^2}$ is a constant. Whenever $e_x = v_x$ and $e_y = v_y$, i.e. when the viewpoint is directly above (below) the delta segment, the projection is zero, and the triangles are coalesced. The probability of satisfying the inequality decreases as e_z approaches v_z, or when the delta segment is viewed from the side. Intuitively, this makes sense–less detail is required for a top-down view of the mesh (assuming a monoscopic view), while more detail is necessary to accurately retain contours and silhouettes in side views. The geometric interpretation of Equation 3 is a solid torus centered at $\mathbf{v}$, with major and minor radii $\frac{n\lambda\delta}{\tau}$. The internal representation and computation of the delta values are further discussed in Section 6.

5 Levels of Detail

In order to allow for efficient view culling and level of detail selection, the mesh is broken up into rectangular blocks. In an application of the algorithm described here, these blocks are represented by a quadtree division of the mesh, although other blocking schemes may be used. It will be shown in the following sections that a gross simplification made on a per block basis can significantly reduce the amount of work required in the fine-grained, per polygon/vertex simplification described in Section 4. The term *level of detail* has been used somewhat loosely up until this point, but will be given a more precise meaning in Section 5.2. In order to maintain a coherent mesh, further restrictions must be put on the simplification condition introduced in the previous section. Within and across block boundaries, a network of dependencies between vertices exists that describes the limits of where triangle fusion can occur.

5.1 Dependencies

As pointed out in Section 4, triangle fusion can occur only when the triangles in the triangle pair appear on the same level in the triangle subdivision. For example, in Figure 1, $\triangle_{a_l a_r}$ and $\triangle_{b_l b_r}$ cannot be coalesced unless the triangles in both pairs (a_l, a_r) and (b_l, b_r) have been fused. It can easily be seen that the triangle pairs represent binary trees, where the smallest triangles correspond to terminal nodes, while coalesced triangles correspond to higher level, nonterminal nodes (hence the subscripts l and r for "left" and "right"). For example, $\triangle_{a_l}$ is a terminal node, $\triangle_{a_l a_r} = (a_l, a_r)$ is a two level tree where the root node is a triangle formed by fusing $\triangle_{a_l}$ and $\triangle_{a_r}$, and $((a_l, a_r), (b_l, b_r))$ is an example of a three level tree.

Another way of looking at triangle fusion is vertex removal, i.e. when two triangles are fused, one vertex

is removed. We call this vertex the *base vertex* of the triangle pair. If the projected delta segment of a base vertex exceeds the threshold τ, we say that the vertex is *activated*; if a vertex is removed, we say that it is *disabled*, or that its *enabled* flag is false. Every triangle pair has a *co-pair* associated with it,[4] and the pair/co-pair share the same base vertex. Hence, each base vertex corresponds to two nodes in two separate, but interlocking, binary vertex trees (see Figure 3). In Figure 1, the pairs (j_l, j_r) and (k_l, k_r) share the base vertex that is incident with all four triangles. A base vertex may be removed only if all base vertices in the four subtrees have been removed. This implies that the *enabled* attribute of a vertex depends on its *activated* attribute, as well as the *enabled* attributes of the children in trees T_1 and T_2, that is

$$\begin{aligned} &activated(v) \wedge \\ &\quad enabled(child_{l1}(v)) \wedge \\ &\quad enabled(child_{r1}(v)) \wedge \\ &\quad enabled(child_{l2}(v)) \wedge \\ &\quad enabled(child_{r2}(v)) \Rightarrow enabled(v). \end{aligned} \tag{4}$$

Figure 3a-f show the dependency relations between vertices. Figure 3i illustrates the tree structure, where tree intersections have been separated for clarity. Figure 3h shows the influence of an activated vertex over other vertices that directly or indirectly depend on it.

To satisfy continuity condition (ii) (see Section 3.1), the algorithm must consider dependencies that cross block boundaries. Since the vertices on block boundaries are shared between adjacent blocks, one must ensure that such shared vertices are referenced uniquely, so that the dependencies may propagate across the boundaries. In most implementations, such shared vertices are simply duplicated, and these redundancies must be resolved before or during the simplification stage. One way of approaching this is to access each vertex via a pointer, and discard the redundant copies of the vertex when a block is read, e.g. during the paging process from disk to memory. Another approach is to ensure that the attributes of all copies of a vertex are kept consistent when updates (e.g. *enabled* and *activated* transitions) occur, which is somewhat similar to the way cache update protocols work. This could be achieved by maintaining a circular linked list of copies for each vertex.

[4] Triangle pairs with base vertices on the edges of the finite data set are an exception.

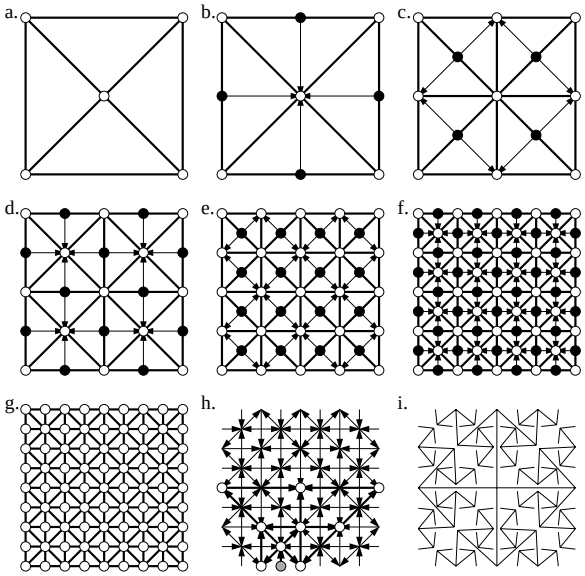

Figure 3: Vertex dependencies. An arc from A to B indicates that B depends on A. The middle bottom figure shows the chain of dependencies from the vertex shaded in grey. The bottom-right figure shows the four binary vertex trees rooted at the center vertex.

5.2 Levels of Evaluation

Complex data sets may consist of millions of polygons, and it is clearly infeasible to run the simplification process on all polygons for each individual frame. One would like to be able to split the simplification process up in two phases; a coarse-grained simplification which determines which discrete levels of detail are needed for each block, and a fine-grained simplification within each block. By obtaining a rough estimate of which vertices can be eliminated in a block, one can often decimate the data by several factors with little computational cost. For example, if one immediately knew that all the solid vertices in Figures 3e and 3f could be discarded, a different level of detail consisting only of the remaining vertices would be used. We call these decimated vertices the block's *lowest level vertices*. If the resulting block's lowest level vertices can likewise be discarded, an even lower level of detail would be used, and a large number of evaluations in the simplification process could be excluded.

We define consecutive levels of detail by decimating every other column and row of the next higher level of detail. The coarse evaluation can be done by computing the maximum delta value for the potentially decimated vertices for each block. Given the *bounding box* (the smallest, axis-aligned rectangular volume that en-

closes the block) of a block and this maximum value, one can determine whether any of these vertices have delta values large enough to exceed τ for a given viewpoint. If none of them do, a lower level of detail may be used. We can expand on this idea to obtain at a more efficient simplification algorithm. By using τ, the view parameters, and the bounding box, one can compute the smallest delta value δ_l that, when projected, can exceed τ, as well as the largest delta value δ_h that may project smaller than τ. Delta values between these extremes fall in an *uncertainty interval*, which we denote by $I_u = [\delta_l, \delta_h]$, for which Equation 3 has to be evaluated. Vertices with delta values less than δ_l can readily be discarded without further evaluation, and conversely, vertices with delta values larger than δ_h cannot be removed. It would obviously be very costly to compute I_u by reversing the projection to get the delta value whose projection equals τ for every single vertex within the block, but one can approximate I_u by assuming that the vertices are dense in the bounding box of the block and, thus, obtain a slightly larger superset of I_u. From here on, we will use I_u to denote this superset.

To find the lower bound δ_l of I_u, the point in the bounding box that maximizes the delta projection must be found. From Equation 1, define $r = \sqrt{(e_x \Leftrightarrow v_x)^2 + (e_y \Leftrightarrow v_y)^2}$ and $h = e_z \Leftrightarrow v_z$. We seek to maximize the function $f(r,h) = \frac{r}{r^2+h^2}$ subject to the constraint $\mathbf{v} \in B$, where B is the set of points contained in the bounding box, described by the two vectors

$$\begin{aligned} \mathbf{b}_{min} &= \left[\begin{array}{ccc} b_{min_x} & b_{min_y} & b_{min_z} \end{array}\right] \\ \mathbf{b}_{max} &= \left[\begin{array}{ccc} b_{max_x} & b_{max_y} & b_{max_z} \end{array}\right]. \end{aligned}$$

Clearly, h^2 has to be minimized, which is done by setting $h = |e_z \Leftrightarrow \text{clamp}(b_{min_z}, e_z, b_{max_z})|$, where

$$\text{clamp}(x_{min}, x, x_{max}) = \begin{cases} x_{min} & \text{if } x < x_{min} \\ x_{max} & \text{if } x > x_{max} \\ x & \text{otherwise} \end{cases}.$$

In the x-y plane, define r_{min} to be the smallest distance from $\left[\begin{array}{cc} e_x & e_y \end{array}\right]$ to the rectangular slice (including the interior) of the bounding box defined by $\left[\begin{array}{cc} b_{min_x} & b_{min_y} \end{array}\right]$ and $\left[\begin{array}{cc} b_{max_x} & b_{max_y} \end{array}\right]$, and define r_{max} to be the largest such distance in the x-y plane. Via partial differentiation with respect to r, the maximum f_{max} of $f(r,h)$ is found at $r = h$. If no $\mathbf{v}$ exists under the given constraints that satisfies $r = h$, r is increased/decreased until $\mathbf{v} \in B$, i.e. $r = \text{clamp}(r_{min}, h, r_{max})$.

The upper bound, δ_h, is similarly found by minimizing $f(r,h)$. This is accomplished by setting $h = \max\{|e_z \Leftrightarrow b_{min_z}|, |e_z \Leftrightarrow b_{max_z}|\}$. f_{min} is then found when either $r = r_{min}$ or $r = r_{max}$, whichever yields a smaller $f(r,h)$.

It is worth pointing out that r and h must always satisfy the condition $r^2 + h^2 \geq n^2$, where n is the same as in Section 4. Hence, r must be chosen such that $r \geq \sqrt{n^2 \Leftrightarrow h^2}$ holds whenever $h < n$.

The bounds on I_u can now be found using the following equations:

$$\delta_l = \left\lceil \frac{2\tau}{n\lambda f_{max}} \right\rceil \tag{5}$$

$$\delta_h = \begin{cases} \left\lfloor \frac{2\tau}{n\lambda f_{min}} \right\rfloor & \text{if } f_{min} > 0 \\ \delta_{max} & \text{if } f_{min} = 0 \end{cases} \tag{6}$$

After computation of I_u, the maximum delta value, δ_{sup},[5] for the lowest level vertices of the block is compared to δ_l, and if smaller, a lower resolution level of detail block is substituted, and the process is repeated for this block. If $\delta_{sup} \geq \delta_l$, it may be that a higher resolution block is needed. By maintaining $\delta^*_{sup} = \max_i\{\delta_{sup_i}\}$, the largest δ_{sup} of all higher resolution blocks (or *block descendants*) for the given area, δ^*_{sup} is compared to δ_l for the current block, and if greater, four higher resolution blocks replace the current block. This implicit hierarchical organization of blocks is best represented as a quadtree, where each block corresponds to a quadnode.

6 Data Structures

Many of the issues related to the data structures used in this algorithm have purposely been left open, as different needs may demand totally different approaches to their representations. In one implementation, as few as six bytes per vertex were used, and as many as 28 bytes were needed to achieve the same goal in another. There are, however, a small number of data structures, suggested here, that will be common to most implementations. As discussed in Section 5, each vertex possesses a number of attributes in addition to the discrete elevation value. One such attribute is the delta value. Assuming 16-bit, unsigned elevation values, the range of delta values becomes $[\delta_{min}, \delta_{max}] = [0, |2z_{max} \Leftrightarrow (z_{min} + z_{min})|] = [0, 131070]$. However, height fields are in general fairly smooth, and from experience, δ_{max} seldom exceeds $\sqrt{z_{max}}$. Hence, we have opted to use eight bits for the delta values, giving a δ_{max} of 255. Delta values that exceed 255 must,

[5] The subscript *sup* stands for *supremum*; the largest member of a closed set.

however, be representable. To accurately represent the frequently occurring, small delta values, while allowing for occasional, larger delta values, a non-linear mapping $c : [0, z_{max}] \rightarrow [0, \delta_{max}]$ that we call the *delta compression function* is applied. (Delta values larger than z_{max} are so rare that we have chosen not to consider them.) We first consider the *decompression function* c^{-1},[6] which should satisfy $c^{-1}(0) = 0$ and $c^{-1}(\delta_{max}) = c^{-1}(255) = 65535 = z_{max}$, and for small x, we want $c^{-1}(x) \simeq x$. Another requirement is $x \leq c^{-1}(c(x)) < c^{-1}(c(y)) = y$ for some $y > x$. The function

$$c^{-1}(x) = \lfloor (x+1)^{1+x^2/255^2} \Leftrightarrow 1 \rfloor \qquad (7)$$

satisfies all of these properties. Figure 4 shows the graph of c^{-1}. The compression function c can then be derived from c^{-1} and the above constraints. Both functions are, for performance reasons, implemented as lookup tables.

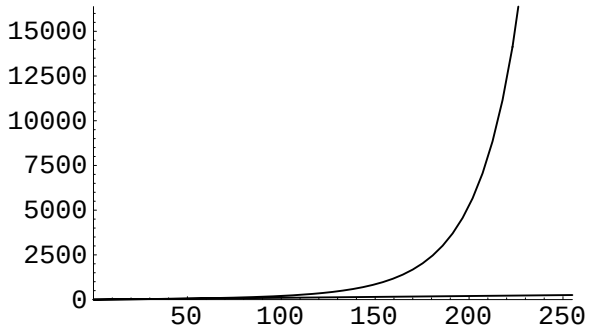

Figure 4: The delta decompression function $c^{-1}(x)$ and $f(x) = x$.

In addition to the delta value, a number of flags must be maintained for each vertex (see Section 5). The flags *enabled* and *activated*, as well as four dependency bits are stored with the delta value (see Figures 3d and 3e). These dependency bits reflect the values of the *enabled* bits of the four "children" (if any) of the vertex. Because the *enabled* bits do not change very frequently, these redundant dependency bits are updated only when a child's *enabled* bit is changed, leading to more efficient accesses in the evaluation of the *enabled* value, and also an arbitrary order of evaluations. The *enabled* flag can be hardwired to either **true** or **false** by setting the *locked* bit of the vertex. This may be necessary, for example, when eliminating gaps between adjacent blocks if compatible levels of detail do not exist, i.e. some vertices on the boundaries of the higher resolution block may have to be disabled. The vertex data structure is shown in Figure 5.

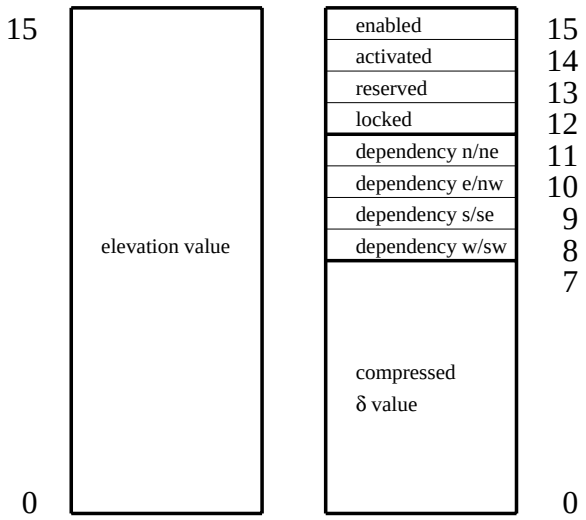

Figure 5: The 32-bit vertex data structure. The bit *reserved* has been reserved for future use.

In the previous section, we derived the uncertainty interval I_u. It was here implied that vertices within ranges of delta values could immediately be accessed. To allow such accesses, the vertices must be sorted on their delta values. However, provision for instantaneous, spatial access to the vertices is required by tasks such as rendering and surface following. This is accomplished by creating an array of indices, where the entries are sorted on the corresponding vertices' delta values. Each entry uniquely references the corresponding vertex via an index pair (i, j) into the 2D array of vertex structures.[7] For each possible delta value, there is a pointer (index) p_δ to a bin that contains the indices to the vertices having that delta value. The 256 bins are stored in ascending order in a contiguous, one-dimensional array. The entries in bin i are then indexed by $p_i, p_i + 1, \ldots, p_{i+1} \Leftrightarrow 1$ ($p_i = p_{i+1}$ implies that bin p_i is empty). Figure 6 illustrates the bin/pointer data structures. The pointers p_i could be represented by 16-bit indices to save memory space.

7 Algorithm Outline

The heart of the algorithm presented here is in the selection of which vertices should be included for rendering

[6] c^{-1} is not the true inverse of c as the compression scheme is lossy.

[7] Alternatively, the vertices could be stored in a 1D, row (column) major array, where only a single index i is needed.

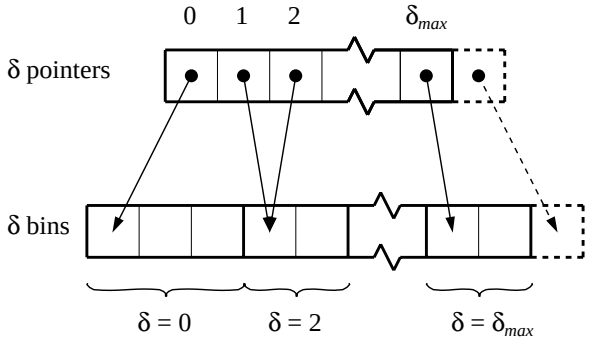

Figure 6: Delta bin data structure. The top, dashed entry is used to delimit the last delta bin.

of the mesh. In Section 7.1, we describe how the mesh is rendered once the vertex selection is done. The vertex selection algorithm can roughly be split up in two parts; level of detail (block) selection, and vertex selection within each block. Using the data structures and equations presented in previous sections, the algorithm is summarized by the pseudo-code below. Unless qualified with superscripts, all variables are assumed to belong to the current frame and block.

```
MAIN()
 1  for each frame n
 2      for each active block b
 3          compute I_u (Equations 5 and 6)
 4          if δ_sup < δ_l
 5              replace b with lower resolution block
 6          else if δ*_sup > δ_l
 7              replace b with higher resolution blocks
 8      for each active block b
 9          determine if b intersects the view frustum
10      for each visible block b
11          I_0 ← [δ_l^{n-1}, δ_l^n ⇔1]
12          I_1 ← [δ_h^n + 1, δ_h^{n-1}]
13          for each vertex v with δ(v) ∈ I_0
14              activated(v) ← false
15              UPDATE-VERTEX(v)
16          for each vertex v with δ(v) ∈ I_1
17              activated(v) ← true
18              UPDATE-VERTEX(v)
19          for each vertex v with δ(v) ∈ I_u
20              EVALUATE-VERTEX(v)
21      for each visible block b
22          RENDER-BLOCK(b)
```

```
UPDATE-VERTEX(v)
1  if ¬locked(v)
2      if ¬dependency_i(v) ∀i
3          if enabled(v) ≠ activated(v)
4              enabled(v) ← ¬enabled(v)
5              NOTIFY(dependent_l(v), dir(l), enabled(v))
6              NOTIFY(dependent_r(v), dir(r), enabled(v))

EVALUATE-VERTEX(v)
1  if ¬locked(v)
2      if ¬dependency_i(v) ∀i
3          activated(v) ← ¬Equation 3
4          if enabled(v) ≠ activated(v)
5              enabled(v) ← ¬enabled(v)
6              NOTIFY(dependent_l(v), dir(l), enabled(v))
7              NOTIFY(dependent_r(v), dir(r), enabled(v))

NOTIFY(v, dir, flag)
1  if v is a valid vertex
2      dependency_dir(v) ← flag
3      if ¬locked(v)
4          if ¬dependency_i(v) ∀i
5              if ¬activated(v)
6                  enabled(v) ← false
7                  NOTIFY(dependent_l(v), dir(l), false)
8                  NOTIFY(dependent_r(v), dir(r), false)
9          else
10             if ¬enabled(v)
11                 enabled(v) ← true
12                 NOTIFY(dependent_l(v), dir(l), true)
13                 NOTIFY(dependent_r(v), dir(r), true)
```

The term *active block* refers to whether the block is currently the chosen level of detail for the area it covers. All blocks initially have I_u set to $[0, \delta_{max}]$, and so do blocks that previously were inactive or outside the field of view. When deactivating vertices with delta values smaller than δ_l, the interval $I_0 \subseteq [0, \delta_l \Leftrightarrow 1]$ is traversed as vertices with smaller delta must have been deactivated in previous frames. Similarly, I_1 is used for vertex activation. In quadtree implementations, the condition on line 4 in MAIN may have to be aggregated; the condition $\delta_{sup} < \delta_l$ should hold for all siblings of b before b can be replaced.

If a vertex's *enable* attribute changes, all dependent vertices must be notified of this change so that their corresponding *dependency* bits are kept consistent with this change. The procedure UPDATE-VERTEX checks if $enabled(v)$ has changed, and if so, notifies v's dependents by calling NOTIFY. If the *enabled* bit of a dependent in turn is modified, NOTIFY is called recursively. Since line 2

in NOTIFY necessarily involves a change of a *dependency* bit, there may be a transition in $enable(v)$ from **true** to **false** on line 6 provided $activate(v)$ is **false** as the vertex is no longer dependent. The evaluation of Equation 3 on line 3 in EVALUATE-VERTEX can be deferred if any of the vertex's dependency bits are set. Note that there may be a one-frame delay before the *activate* attribute is corrected due to this deferral if the child vertices are evaluated after the dependent (see line 2 of EVALUATE-VERTEX and lines 4-5 of NOTIFY). The function $dir(x)$ associated with a vertex v refers to the bit of dependent x that reflects the *enabled* bit of vertex v. Note that a check has to be made (line 1 in NOTIFY) whether a vertex is "valid" as some vertices have fewer than two dependents (e.g. corner and center vertices in a block).

7.1 Mesh Rendering

Once the vertex selection is made, a coherent triangle mesh must be formed that connects the selected vertices. This mesh is defined by recursively bisecting right triangles,[8] stopping when the *enabled* attribute of a base vertex is **false**. To efficiently render the mesh, a *triangle mesh graphics primitive*, such as the one supported by IRIS GL and OpenGL [11, 20], may be used. For each specified vertex v, the previous two vertices and v form the next triangle in the mesh. At certain points, the previous two vertices must be swapped via a `swaptmesh()` call (IRIS GL), or a `v3f()` call (OpenGL). The two-entry *vertex stack* is maintained explicitly to allow the decision as to when to swap the stack to be made. The following pseudo-code describes the mesh rendering algorithm. This procedure is called four times by RENDER-BLOCK with two of the corner vertices v_{i_l} and v_{i_r}, and the center vertex v_{i_t} of the block. In the top level invocation, *level* is $2n$, where $2^n + 1$ is the block size, and *post-swap* is **false**.

```
RENDER-TMESH(i_l, i_t, i_r, level, post-swap)
 1  if level > 0
 2      if enabled(v_{i_t})
 3          i_b ← (i_l + i_r)/2
 4          if i_t = bottom(my-stack)
 5              swap top and bottom of my-stack
 6              do-swap ← ¬do-swap
 7          RENDER-TMESH(i_l, i_b, i_t, level ⇔1, false)
 8          if i_t ≠ top(my-stack) ∧ i_t ≠ bottom(my-stack)
 9              if do-swap
10                  swap graphics stack
11                  do-swap ← false
12              render vertex v_{i_t}
13              bottom(my-stack) ← top(my-stack)
14              top(my-stack) ← i_t
15          RENDER-TMESH(i_t, i_b, i_r, level ⇔1, true)
16          if post-swap
17              swap top and bottom of my-stack
18              do-swap ← ¬do-swap
```

In RENDER-BLOCK, one of the corner vertices must first be rendered and put on *my-stack*, and after RENDER-TMESH is called, the other corner vertex must be rendered to close the mesh. i_b is the index of the (base) vertex that in the x-y plane is the midpoint of the edge $\overline{v_{i_l} v_{i_r}}$. Since *my-stack* reflects what vertices are currently on the graphics stack, line 10 could be implemented with a `v3f()`, using the vertex indexed by $bottom(my\text{-}stack)$.

8 Results

To show the effectiveness of the polygon reduction algorithm, we here include graphs of number of polygons and delta projections, frame rates, and error percentages in the images produced. A set of color plates illustrate wireframe triangulations, textured terrain, and difference images that highlight the varying image quality resulting from different choices of τ. The height field data used comes from a 2×2 meter uniform resolution digital elevation model of the Hunter-Liggett military base in California, with discrete elevation values at one meter height resolution. All 24-bit images were generated on a two-processor, 150 MHz SGI Onyx RealityEngine^2 [1], and have dimensions 1024×768 pixels unless otherwise specified.

We first examine the amount of polygon reduction as a function of the threshold τ. A typical view of the terrain, showing a variety of features such as ridges, valleys, bumps, and relatively flat areas, was chosen for this purpose. Figure 7 shows four curves drawn on a logarithmic scale (y-axis). The top vertical line shows the total number of polygons in the view frustum before any reduction method is applied. This number is approximately 23 million. The curve second from the top represents the number of polygons remaining after the block-based level of detail selection is done. We will use the data for this curve as a representative case of multi-resolution regular grid algorithms, and compare with our algorithm. This is a fair comparison as the maximum error in the surface geometry is the same for both algorithms. The number of polygons rendered, i.e. remaining polygons after the

[8] This assumes that the triangles are viewed in the x-y plane, discarding the height component. In three dimensions, the triangles may not be right, neither are they truly bisected.

vertex-based simplification, is shown by the lowest solid curve. As expected, these two curves flatten out as τ is increased. The ratio of the total number of polygons and the number of rendered polygons ranges from about five ($\tau = 0$) to a little over 3,000 ($\tau = 4$). Of course, at $\tau = 0$, only coplanar triangles are fused. The ratio of the middle two curves varies between 5 and 55, giving a significant advantage to our algorithm as τ is increased. We pay special attention to the data obtained at $\tau = 1$, as this threshold is small enough that virtually no popping can be seen in animated sequences, and for all purposes, the resulting images are virtually identical to the ones obtained with no simplification. Color plates 1a–c illustrate the three cases of simplification at $\tau = 1$. In Color Plate 1c, note how many polygons are required for the high frequency data, while only a few, large polygons are used for the flatter areas. For this particular threshold, the ratio between the number of polygons before simplification and the number of polygons after vertex-based simplification is 343, while the ratio between block-based and vertex-based simplification is 17. The bottommost, dashed curve in Figure 7 represents the total number of delta values that fall in the uncertainty interval per frame (Section 5.2). Note that this quantity is generally an order of magnitude smaller than the number of rendered polygons. This is significant as the evaluations associated with these delta values constitute the bulk of the computation in terms of CPU time. This also shows the advantage of computing the uncertainty interval–out of the 11.5 million vertices contained in the view frustum, only 14,000 evaluations of Equation 3 need to be made when $\tau = 1$.

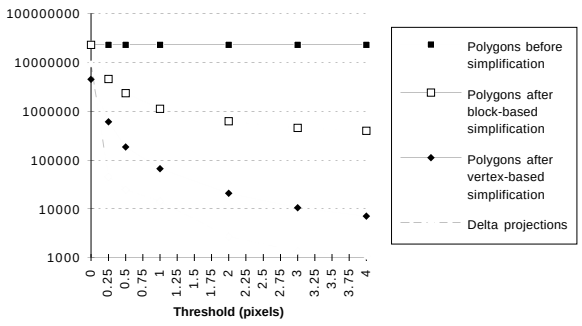

Figure 7: The number of polygons as a function of τ. The bottom curve shows the number of times Equation 3 was evaluated per frame.

In order to evaluate the errors due to the simplification, the height field was textured using both a black and white checkerboard pattern (Color plates 2a–c), and real photoimagery (Color Plates 3a–c). Note that no simplification algorithm has been applied to the texture. Again, the threshold is set to one pixel. The high contrast in the checkerboard pattern makes identification and interpretation of the errors easy in a difference image, although the rendered images look virtually identical. To quantitatively measure the errors, all pixels that do not match perfectly in the checkerboard images are counted and assigned a unique color, leaving the remaining pixels transparent, and are then superimposed on top of the vertex-based simplification wireframe images (see Color Plates 4a–f). Figure 8 shows the percentage of non-matching pixels as τ is varied. In the case of $\tau = 1$, the percentage of non-matching pixels is roughly 5%. Note that when $\tau = 0$, the percentage should theoretically be zero, but is in fact 0.25%. This is due to aliasing and the finite resolution supported by the z-buffer, as many distant polygons render into the same screen pixel. If anti-aliasing were done, the errors in the images where τ is large would be relatively smaller. A quantitative analysis of the aliasing effects has not yet been done. In the current implementation of the algorithm that was used for image generation, cross block dependencies are ignored, leaving occasional small gaps on the boundaries between blocks when τ exceeds one pixel. In the images presented here, such gaps have been filled by vertical polygons.

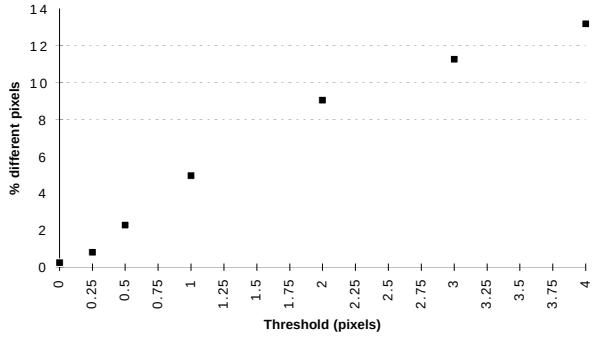

Figure 8: Percentage of erroneous pixels in vertex-based simplification checkerboard image.

Figure 9 shows how the quantities in Figure 7, as well as the frame rate vary with time. The data collection for 3,230 frames was done over a time period of 120 seconds, with the viewpoint following a circular path of radius 1 km. The terrain was rendered as a wireframe mesh in a 640×480 window, with $\tau = 2$ pixels. It can be seen that the number of rendered polygons does not depend on the total number of polygons in the view frustum–as a matter of fact, the number of rendered polygons is lowest at the point when the number of visible polygons is close to its

maximum. The number of rendered polygons depends entirely on the complexity of the terrain intersected by the view frustum. As evidenced by the graph, a frame rate of at least 20 frames per second was sustained throughout the two minutes of fly-through.

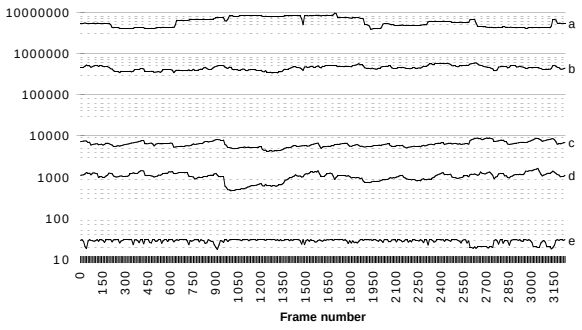

Figure 9: Time graph of (a) total number of polygons in view frustum, (b) number of polygons after block-based simplification, (c) number of polygons after vertex-based simplification, (d) number of delta projections, and (e) frames per second.

9 Conclusion

We have shown that the algorithm presented in this paper, which is based on real-time, per polygon, level of detail evaluation, can achieve interactive and consistent frame rates exceeding twenty frames per second, with only a minor loss in image quality. A polygon reduction of the original data of a factor of 300 can be made with errors below 5% in the resulting image. Compared to multiresolution, regular grid renderings of equal accuracy, this algorithm generally performs more than ten times better in terms of the number of rendered polygons. The concept of continuous level of detail allows a polygon distribution that is near optimal for any given viewpoint and frame, and also yields smooth changes in the number of rendered polygons. A single parameter that can be changed interactively, with no incurred cost, determines the resulting image quality, and a relationship between this parameter and the number of rendered polygons exists, providing capabilities for frame rate maintenance. Attractive features attributed to regular grid representations, such as fast geometric queries, compact representation, and fast mesh rendering are retained. With little extra effort, the algorithm can be extended to handle the problem of gaps between blocks of different levels of detail, as well as geometry blending to allow further polygon reduction with elimination of popping effects.

References

[1] Kurt Akeley. RealityEngine graphics. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 27:109–116, August 1993.

[2] Michael A. Cosman, Allen E. Mathisen, and John A. Robinson. A new visual system to support advanced requirements. In *Proceedings, IMAGE V Conference*, pages 370–380, June 1990.

[3] Leila De Floriani. A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics and Applications*, 9(2):67–78, March 1989.

[4] David H. Douglas. Experiments to locate ridges and channels to create a new type of digital elevation model. *Cartographica*, 23(4)29–61, 1986.

[5] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. *Computer Graphics (SIGGRAPH '95 Proceedings)*, 29:173–182, August 1995.

[6] John S. Falby, Michael J. Zyda, David R. Pratt, and Randy L. Mackey. NPSNET: Hierarchical data structures for real-time three-dimensional visual simulation. *Computers & Graphics*, 17(1):65–69, 1993.

[7] R.L. Ferguson, R. Economy, W.A. Kelly, and P.P. Ramos. Continuous terrain level of detail for visual simulation. In *Proceedings, IMAGE V Conference*, pages 144–151, June 1990.

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

[9] T.A. Funkhouser and C.H. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 27:247–254, August 1993.

[10] Michael Garland and Paul S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, CS Dept., Carnegie Mellon U., 1995.

[11] *Graphics Library Programming Guide*, Silicon Graphics Computer Systems, Mountain View, Calif., 1991.

[12] M.H. Gross, R. Gatti, and O. Staadt. Fast multiresolution surface meshing. In *Proceedings of Visualization '95*, pages 135–142, October 1995.

[13] Paul S. Heckbert and Michael Garland. Multiresolution modeling for fast rendering. In *Proceedings of Graphics Interface '94*, pages 1–8, 1994.

[14] Paul S. Heckbert and Michael Garland. Survey of surface approximation algorithms. Technical Report CMU-CS-95-194, CS Dept., Carnegie Mellon U., 1995.

[15] J. Rohlf and J. Helman. IRIS Performer: a high performance multiprocessing toolkit for real-time 3D graphics. *Computer Graphics (SIGGRAPH '94 Proceedings)*, 28(3):381–394, July 1994.

[16] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[17] Lori L. Scarlatos. A refined triangulation hierarchy for multiple levels of terrain detail. In *Proceedings, IMAGE V Conference*, pages 114–122, June 1990.

[18] Florian Schroder and Patrick Rossbach. Managing the complexity of digital terrain models. *Computers & Graphics*, 18(6):775–783, 1994.

[19] William J. Schroeder, Jonathon A. Zarge, and William E. Lorenson. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):65–70, July 1992.

[20] Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification*, Silicon Graphics, Inc., 1992.

[21] David A. Southard. Piecewise planar surface models from sampled data. *Scientific Visualization of Physical Phenomena*, pages 667-680, June 1991.

[22] David C. Taylor and William A. Barret. An algorithm for continuous resolution polygonalizations of a discrete surface. In *Proceedings of Graphics Interface '94*, pages 33–42, 1994.