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

Peter Lindstrom* David Koller* William Ribarsky*
Larry F. Hodges* Nick Faust† Gregory A. Turner‡

*† Georgia Institute of Technology

‡ SAIC

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. A coarse level of simplification is performed to select discrete levels of detail for blocks of the surface mesh, followed by further simplification through repolygonalization in which individual mesh vertices are considered for removal. These steps compute and generate the appropriate level of detail dynamically in real-time, minimizing the number of rendered polygons and allowing for smooth changes in 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.

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.

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

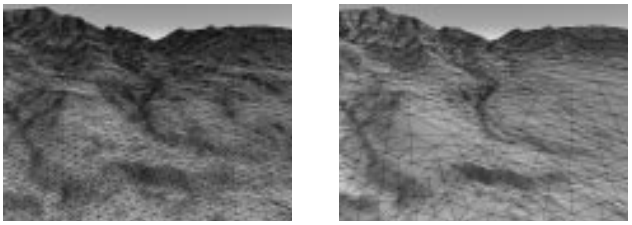


Figure 1: Terrain surface tessellations corresponding to projected geometric error thresholds of one (left) and four (right) pixels.

In this paper we present a new level of detail display algorithm that is applicable to surfaces that are represented as uniformly-gridded polygonal height fields. By extending the regular grid representation to allow polygons to be recursively combined where appropriate, a mesh with fewer polygons can be used to represent the height field (Figure 1). Such small, incremental changes to the mesh polygonalization provide for continuous levels of detail and a near optimal tessellation for any given viewpoint. The algorithm is characterized by the following set of features:

- **Large reduction in the number of polygons to be rendered.** Typically, the surface grid is decimated by several orders of magnitude with no or little loss in image quality, accommodating interactive frames rates for smooth animation.
- **Smooth, continuous changes between different surface levels of detail.** The number and distribution of rendered polygons change smoothly between successive frames, affording maintenance of consistent frame rates.
- **Dynamic generation of levels of detail in real-time.** The need for expensive generation of multiresolution models ahead of time is eliminated, allowing dynamic changes to the surface geometry to be made with little computational cost.
- **Support for 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.

Related approaches to polygonal surface approximation and multiresolution rendering are discussed in the next section. The following sections of the paper describe the theory and procedures necessary 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.

*Graphics, Visualization, & Usability Center, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332–0280. {lindstro, koller, ribarsky, hodges}@cc.gatech.edu.

†Center for GIS and Spatial Analysis Technologies, Georgia Tech Research Institute. nick.faust@gtri.gatech.edu.

‡Simulation Technology Division. gturner@std.saic.com.

2 RELATED WORK

A large number of researchers have developed algorithms for approximating terrains and other height fields using polygonal meshes. These algorithms attempt to represent surfaces with a given number of vertices, or within a given geometric error metric, or in a manner that preserves application specific critical features of the surface. Uniform grid methods or irregular triangulations are employed to represent the surfaces, and techniques including hierarchical subdivisions and decimations of the mesh are used for simplification and creation of multiresolution representations.

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 [9, 10, 19], and hierarchical triangulation representations have been proposed that lend themselves to usage in level of detail algorithms [3, 4, 18]. 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. 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, 7]. Such uniform polygonalizations generally produce many more polygons than TINs for a given level of approximation, but grid representations are 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 methods that meet application specific criteria. Fowler and Little [9] construct TINs characterized by certain "surface specific" points and critical lines, allowing the TIN representation to closely match important terrain features. Douglas [5] locates specific terrain features such as ridges and channels in a terrain model database, and represents the surface with line segments from these "information rich" features. 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. This technique, however, has not yet proven to yield interactive frame rates. The general problem of surface simplification has been addressed with methods for mesh decimation and optimization [14, 20], although these techniques are not suitable for on-the-fly generation of multiple levels 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 [8] claims that such blending techniques between 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 resolution.

3 MOTIVATION

The algorithm presented in this paper has been designed to meet a number of criteria desirable for a real-time level of detail (LOD) algorithm for height fields. These characteristics include:

(i) At any instant, the mesh geometry and the components that describe it should be directly and efficiently queryable, allowing for surface following and fast spatial indexing of both polygons and vertices.

(ii) Dynamic changes to the geometry of the mesh, leading to recomputation of surface parameters or geometry, should not significantly impact the performance of the system.

(iii) High frequency data such as localized convexities and concavities, and/or local changes to the geometry, should not have a widespread 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 direct 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, a polygon-based level of detail algorithm that supports all of these features is clearly of great importance in areas such as terrain rendering, which often requires both high frame rates and high visual fidelity, as well as fast and frequent queries of a possibly deformable terrain surface. Our algorithm successfully achieves all of the goals listed above.

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 requires extensive computational effort. 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 dynamic deformations of the mesh occur, e.g. in the form of explosions.

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 coarse level of detail selection.) 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 [16, 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 substantial fraction of the given rendering capacity, and may cause rapid fluctuations in the frame rate.

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 characterize the data with a small set of parameters that are used in conjunction with viewpoint distance and view angle to select what could be considered "appropriate" levels of detail. Examples of such algorithms include TIN simplification [9], feature (e.g. peaks, ridges, and valleys) identification and preservation [5, 21], and frequency analysis/transforms such as wavelet simplification [6, 12]. These algorithms often do not provide enough information to derive a tight bound on the maximum error in the projected image. 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 include: flexibility and efficiency afforded by a regular grid representation; localized polygon densities due to variable resolution within each block; screen-space error-driven LOD selection determined by a single threshold; 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 [8, 18, 22], some of which are discussed below. As mentioned in (iii) and (iv) above, it is important that the complexity of the surface geometry 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 multiresolution height fields, 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 morphs (blends) 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 over distance from the viewpoint to the mesh.

(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 resolution 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 polygon distribution function $n(\mathbf{v},A)$. For any given area $A \subseteq \mathbf{R}^2$, the number of polygons used to describe the area is continuous with respect to the viewpoint $\mathbf{v}$.[1] Note that A does not necessarily have to be a connected set. Since the image of n is discrete, we define continuity in terms of the *modulus of continuity* $\omega(\delta,n)$. We say that n is continuous iff $\omega(\delta,n) \rightarrow \varepsilon$, for some $\varepsilon \leq 1$, as $\delta \rightarrow 0$. That is, for sufficiently small changes in the viewpoint, the change in the number of polygons over A is at most one. As a consequence of a continuous polygon distribution, the number of rendered polygons (after clipping), $n(\mathbf{v})$, is continuous with respect to the viewpoint.

Note that a continuous level of detail algorithm may possess one or more of these independent properties (e.g. (i) does not in general imply (iii), and vice versa). Depending on the constraints inherent in the tessellation method, criterion (iii) may or may not be satisfiable, but a small upper bound ε_{max} on ε may exist. Our algorithm, as presented here, primarily addresses definition (iii), but has been designed to be easily extensible to cover the other two definitions (the color plates included in this paper reflect an implementation satisfying (ii)).

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

4 SIMPLIFICATION CRITERIA

The surface simplification process presented here is best described as a sequence of two steps: a coarse-grained simplification of the height field mesh geometry that is done to determine which discrete level of detail models are needed, followed by a fine-grained retriangulation of each LOD model in which individual vertices are considered for removal. The algorithm ensures that no errors are introduced in the coarse simplification beyond those that would be introduced if the fine-grained simplification were applied to the entire mesh. Both steps are executed for each rendered frame, and all evaluations involved in the simplification are done dynamically in real-time, based on the location of the viewpoint and the geometry of the height field.

The height field is described by a rectilinear grid of points elevated above the x-y plane, with discrete sampling intervals of x_{res} and y_{res}. The surface corresponding to the height field (before simplification) is represented as a symmetric triangle mesh. The smallest mesh representable using this triangulation, the *primitive mesh*, has dimensions 3×3 vertices, and successively larger meshes are formed by grouping smaller meshes in a 2×2 array configuration (see Figure 2). For any level l in this recursive construction of the mesh, the vertex dimensions x_{dim} and y_{dim} are $2^l + 1$. For a certain level n, the resulting mesh is said to form a *block*, or a discrete level of detail model. A set of such blocks of fixed dimensions $2^n + 1$ vertices squared, describes the height field dataset, where the boundary rows and columns between adjacent blocks are shared. While the dimensions of all blocks are fixed, the spatial extent of the blocks may vary by multiples of powers of two of the height field sampling resolution, i.e. the area of a block is $2^{m+n}x_{res} \times 2^{m+n}y_{res}$ where m is some non-negative integer. Thus, lower resolution blocks can be obtained by discarding every other row and column of four higher resolution blocks. We term these decimated vertices the *lowest level vertices* of a block (see Figure 2c). A *quadtree* data structure [17] naturally lends itself to the block partitioning of the height field dataset described above.

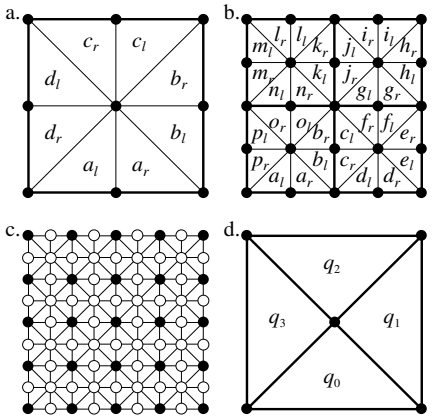


Figure 2: (a, b) Triangulation of uniform height fields of dimensions 3×3 and 5×5 vertices, respectively. (c) Lowest level vertices (unfilled). (d) Block quadrants.

In the following sections, we describe the different simplification steps. We begin by deriving a criterion for the fine-grained (vertex-based) simplification. The coarse-grained (block-based) level of detail selection is then described in terms of the former.

4.1 Vertex-Based Simplification

In the fine-grained simplification step, many smaller triangles are removed and replaced with fewer larger triangles. Conceptually, at the beginning of each rendered frame, the entire height field dataset at its highest resolution is considered. Wherever certain conditions are met, a *triangle/co-triangle pair* $(\triangle_{a_l}, \triangle_{a_r})$ is reduced to one single triangle $\triangle_{a_l} \oplus \triangle_{a_r}$, and the resulting triangle and its co-triangle (if one exists) are considered for further simplification in a recursive manner. In the x-y plane with $x_{res} = y_{res}$, a triangle/co-triangle pair is defined by the two congruent right triangles obtained by bisecting a larger isosceles right triangle. Recursive bisection of the resulting two triangles yields lower level triangle/co-triangle pairs. Triangle/co-triangle pairs within a block are descended from the four triangular quadrants of the block, defined by the block boundary and its diagonals (see Figure 2d). For arbitrary height field resolutions, the square mesh is simply stretched in either dimension while retaining the vertex connections. Figure 2a and 2b illustrate the lowest level pairs, where each pair has been assigned a unique letter.

The conditions under which a triangle pair can be combined into a single triangle are primarily described by the amount of change in slope between the two triangles. For triangles $\triangle ABE$ and $\triangle BCE$, with A, B, and C in a plane perpendicular to the x-y plane, the slope change is measured by the vertical (z axis) distance $\delta_B = |B_z - \frac{A_z+C_z}{2}|$, i.e. the maximum vertical distance between $\triangle ACE = \triangle ABE \oplus \triangle BCE$ and the triangles $\triangle ABE$ and $\triangle BCE$ (see Figure 3). This distance is referred to as vertex B's *delta value*. 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 merged triangle and its corresponding sub-triangles. If this error is smaller than a given threshold, τ, the triangles may be fused. If the resulting triangle has a co-triangle with error smaller than the threshold, this pair is considered for further simplification. This process is applied recursively until no further simplification of the mesh can be made. Note that this scheme typically involves a reduction of an already simplified mesh, and the resulting errors (i.e. the projected delta segments) are not defined with respect to the highest resolution mesh, but rather relative to the result of the previous iteration in the simplification process. However, empirical data indicates that the effects of this approximation are negligible (see Section 7).

We now derive a formula for the length of the projected delta segment. Let $\mathbf{v}$ be the midpoint of the delta segment,[2] and define $\mathbf{v}^+ = \mathbf{v} + \begin{bmatrix} 0 & 0 & \frac{\delta}{2} \end{bmatrix}$, $\mathbf{v}^- = \mathbf{v} - \begin{bmatrix} 0 & 0 & \frac{\delta}{2} \end{bmatrix}$. 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 d 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 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. 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 $-\hat{\mathbf{z}}$. This approximation is reasonable as long as the field of view is relatively small, and its effect is that the projected delta segments that represent the errors in the triangle simplification become relatively smaller at the periphery of the screen, where less detail is then used—an artifact that is often acceptable as human visual perception degrades toward the periphery.

- We assume $v^+_{eye_z} \simeq v^-_{eye_z} \simeq v_{eye_z}$ in the perspective division $\frac{1}{-v_{eye_z}}$. This is a fair assumption because, in general, $\delta \ll ||\mathbf{e}-\mathbf{v}|| = -v_{eye_z}$.

[2] One may safely substitute the vertex associated with the delta segment for its midpoint.

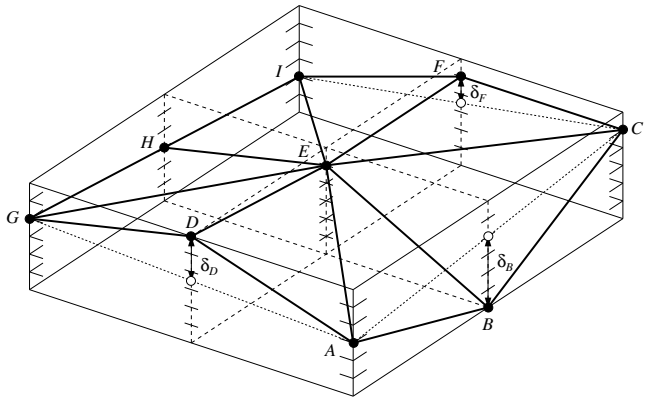


Figure 3: Geometric representation of delta values. $\delta_B = 4$, $\delta_D = 2.5$, $\delta_F = 1.5$, $\delta_H = 0$.

According to the first approximation, the viewing matrix is then:

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

with $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ perpendicular to $\mathbf{e}-\mathbf{v}$ at all times. This definition of $\mathbf{M}$ leads to the following equalities:

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

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

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

For performance reasons, δ^2_{screen} is compared to τ^2 so that the square root can be avoided:

$$\frac{d^2\lambda^2\delta^2\left((e_x-v_x)^2+(e_y-v_y)^2\right)}{\left((e_x-v_x)^2+(e_y-v_y)^2+(e_z-v_z)^2\right)^2} \leq \tau^2$$

An equivalent inequality that defines the simplification condition reduces to a few additions and multiplications:

$$\delta^2\left((e_x-v_x)^2+(e_y-v_y)^2\right) \leq \kappa^2\left((e_x-v_x)^2+(e_y-v_y)^2+(e_z-v_z)^2\right)^2 \quad (2)$$

where $\kappa = \frac{\tau}{d\lambda}$ is a constant. Whenever $e_x = v_x$ and $e_y = v_y$, i.e. when the viewpoint is directly above or 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. This makes sense, intuitively, as 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 the complement of Equation 2 is a "bialy"—a solid circular torus with no center hole—centered at $\mathbf{v}$, with radius $r = \frac{d\lambda\delta}{2\tau}$ (see Figure 4). The triangles associated with $\mathbf{v}$ can be combined provided that the viewpoint is not contained in the bialy.

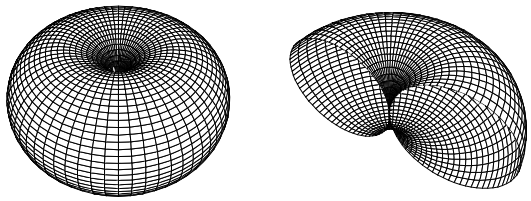


Figure 4: Geometric representation (and its cross-section) of the boundary of Equation 2.

4.2 Block-Based Simplification

Complex datasets may consist of millions of polygons, and it is clearly too computationally expensive to run the simplification process described in the previous section on all polygon vertices for each individual frame. By obtaining a conservative estimate of whether certain groups of vertices can be eliminated in a block, the mesh can often be decimated by several factors with little computational cost. If it is known that the maximum delta projection of all lowest level vertices in a block falls within τ, those vertices can immediately be discarded, and the block can be replaced with a lower resolution block, which in turn is considered for further simplification. Accordingly, a large fraction of the delta projections can be avoided.

The discrete level of detail selection is done by computing the maximum delta value, δ_{max}, of the lowest level vertices for each block. Given the axis-aligned bounding box of a block and δ_{max}, one can determine, for a given viewpoint, whether any of these vertices have delta values large enough to exceed the threshold τ. If none of them do, a lower resolution model may be used. We can expand on this idea to obtain a more efficient simplification algorithm. By using τ, the view parameters, and the constraints provided by 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 2 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 this point 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 $r = \sqrt{(e_x-v_x)^2+(e_y-v_y)^2}$ and $h = e_z - v_z$. We seek to maximize the function $f(r,h) = \frac{r}{r^2+h^2}$ subject to the constraints $r^2+h^2 \geq d^2$ and $\mathbf{v} \in B$, where d is the distance from the viewpoint to the projection plane and B is the set of points contained in the bounding box, described by the two vectors

$$\mathbf{b}_{min} = \begin{bmatrix} b_{min_x} & b_{min_y} & b_{min_z} \end{bmatrix}$$
$$\mathbf{b}_{max} = \begin{bmatrix} b_{max_x} & b_{max_y} & b_{max_z} \end{bmatrix}$$

We solve this optimization problem by constraining r, such that $d^2 - h^2 \leq r^2_{min} \leq r^2 \leq r^2_{max}$ (r and h are otherwise independent). Clearly, then, h^2 has to be minimized which is accomplished by setting $h = h_{min} = |e_z - \text{clamp}(b_{min_z}, e_z, b_{max_z})|$, 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 $\begin{bmatrix} e_x & e_y \end{bmatrix}$ to the rectangular slice (including the interior) of the bounding box defined by $\begin{bmatrix} b_{min_x} & b_{min_y} \end{bmatrix}$ and $\begin{bmatrix} b_{max_x} & b_{max_y} \end{bmatrix}$, and define r_{max} to be the largest such distance. 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 done by setting $h = h_{max} = \max\{|e_z - b_{min_z}|, |e_z - b_{max_z}|\}$. f_{min} is then found when either $r = r_{min}$ or $r = r_{max}$, whichever yields a smaller $f(r,h)$.

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

$$\delta_l = \frac{\tau}{d\lambda f_{max}} \quad (3)$$

$$\delta_h = \begin{cases} 0 & \text{if } \tau = 0 \\ \frac{\tau}{d\lambda f_{min}} & \text{if } \tau > 0 \text{ and } f_{min} > 0 \\ \infty & \text{otherwise} \end{cases} \quad (4)$$

After computation of I_u, δ_{max} 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 $\delta_{max} > \delta_l$, it may be that a higher resolution block is needed. By maintaining $\delta^*_{max} = \max_i\{\delta_{max_i}\}$, the largest δ_{max} of all higher resolution blocks (or *block descendants*) for the given area, δ^*_{max} is compared to δ_l for the current block, and if greater, four higher resolution blocks replace the current block. As mentioned earlier, this implicit hierarchical organization of blocks is best represented by a quadtree, where each block corresponds to a quadnode.

4.3 Vertex Dependencies

As pointed out in Section 4.1, 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 2b, $\triangle_{e_l} \oplus \triangle_{e_r}$ and

$\triangle_{f_l} \oplus \triangle_{f_r}$ cannot be coalesced unless the triangles in both pairs $(\triangle_{e_l}, \triangle_{e_r})$ and $(\triangle_{f_l}, \triangle_{f_r})$ have been fused. The triangles can be represented by nodes in a binary expression tree, where the smallest triangles correspond to terminal nodes, and coalesced triangles correspond to higher level, nonterminal nodes formed by recursive application of the $\oplus$ operator (hence the subscripts l and r for "left" and "right"). Conceptually, this tree spans the entire height field dataset, but can be limited to each block.

Another way of looking at triangle fusion is as vertex removal, i.e. when two triangles are fused, one vertex is removed. We call this vertex the *base vertex* of the triangle pair. Each triangle pair has a *co-pair* associated with it,[3] and the pair/co-pair share the same base vertex. The mapping of vertices to triangle pairs, or the nodes associated with the operators that act on the triangle pairs, results in a *vertex tree*, wherein each vertex occurs exactly twice; once for each triangle pair (Figures 5g and 5h). Hence, each vertex has two distinct parents (or dependents)—one in each of two binary subtrees T_0 and T_1—as well as four distinct children. If any of the descendants of a vertex v are included in the rendered mesh, so is v, and we say that v is *enabled*. If the projected delta segment associated with v exceeds the threshold τ, v is said to be *activated*, which also implies that v is *enabled*. Thus, the *enabled* attribute of v is determined by

$$\begin{aligned} &activated(v) \vee \\ &\quad enabled(left_{T_0}(v)) \vee \\ &\quad enabled(right_{T_0}(v)) \vee \\ &\quad enabled(left_{T_1}(v)) \vee \\ &\quad enabled(right_{T_1}(v)) \Rightarrow enabled(v) \end{aligned}$$

An additional vertex attribute, *locked*, allows the *enabled* flag to be hardwired to either **true** or **false**, overriding the relationship above. 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 permanently disabled. Figures 5a–e show the dependency relations between vertices level by level. Figure 5f shows the influence of an enabled vertex over other vertices that directly or indirectly depend on it. Figures 5g and 5h depict the two possible vertex tree structures within a block, where intersections have been separated for clarity.

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, these vertices must be 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 before the block is first accessed. 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. This can be achieved by maintaining a circular linked list of copies for each vertex.

5 ALGORITHM OUTLINE

The algorithm presented here describes the steps necessary to select which vertices should be included for rendering of the mesh. In Section 5.1, we describe how the mesh is rendered once the vertex selection is done. A discussion of appropriate data structures is presented in Section 6. Using the equations presented in previous sections, the algorithm is summarized by the pseudocode below. Unless qualified with superscripts, all variables are assumed to belong to the current frame and block.

[3] Triangle pairs with base vertices on the edges of the finite dataset are an exception.

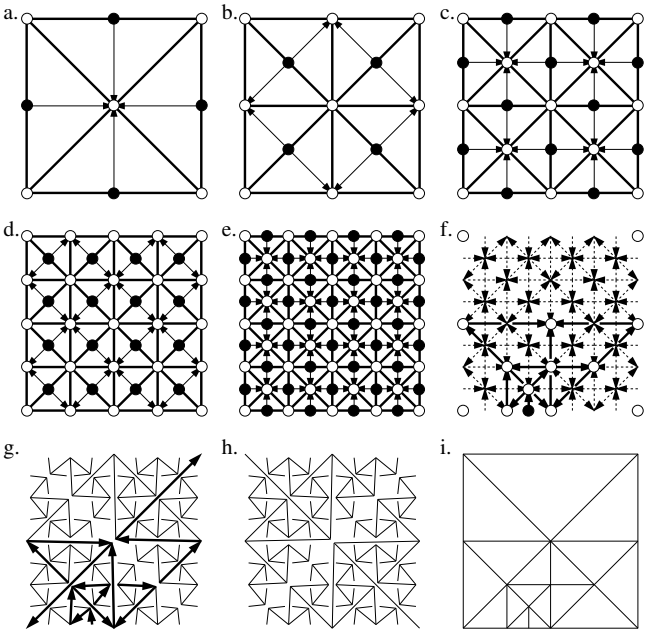


Figure 5: (a–e) Vertex dependencies by descending levels (left to right, top to bottom). An arc from A to B indicates that B depends on A. (f) Chain of dependencies originating from the solid vertex. (g, h) Symmetric binary vertex trees (the arcs in (g) correspond to (f)). (i) Triangulation corresponding to (f).

```
MAIN()
 1  for each frame n
 2      for each active block b
 3          compute I_u (Equations 3 and 4)
 4          if δ_max ≤ δ_l
 5              replace b with lower resolution block
 6          else if δ*_max > 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]
12          I_1 ← (δ_h^n, 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(parent_{T_0}(v), branch_{T_0}(v), enabled(v))
 6              NOTIFY(parent_{T_1}(v), branch_{T_1}(v), enabled(v))
```

```
EVALUATE-VERTEX(v)
 1  if ¬locked(v)
 2      if ¬dependency_i(v) ∀i
 3          activated(v) ← ¬Equation 2
 4          if enabled(v) ≠ activated(v)
 5              enabled(v) ← ¬enabled(v)
 6              NOTIFY(parent_T0(v), branch_T0(v), enabled(v))
 7              NOTIFY(parent_T1(v), branch_T1(v), enabled(v))

NOTIFY(v, child, state)
 1  if v is a valid vertex
 2      dependency_child(v) ← state
 3      if ¬locked(v)
 4          if ¬dependency_i(v) ∀i
 5              if ¬activated(v)
 6                  enabled(v) ← false
 7                  NOTIFY(parent_T0(v), branch_T0(v), false)
 8                  NOTIFY(parent_T1(v), branch_T1(v), false)
 9          else
10              if ¬enabled(v)
11                  enabled(v) ← true
12                  NOTIFY(parent_T0(v), branch_T0(v), true)
13                  NOTIFY(parent_T1(v), branch_T1(v), 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 $[0,\infty)$, and so do blocks that previously were inactive. When deactivating vertices with delta values smaller than δ_l, the interval $I_0 \subseteq [0,\delta_l]$ is traversed. By inductive reasoning, vertices with deltas smaller than the lower bound of I_0 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 supplemented; the condition $\delta_{max} \leq \delta_l$ should also hold for the three neighboring siblings of b before b can be replaced.

If a vertex's *enabled* attribute changes, all dependent vertices must be notified of this change so that their corresponding *dependency* flags 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* flag 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 $enabled(v)$ from **true** to **false** on line 6 provided $activated(v)$ is **false** as the vertex is no longer dependent. The evaluation of Equation 2 on line 3 in EVALUATE-VERTEX can be deferred if any of the vertex's *dependency* flags are set, which is of significant importance as this evaluation is one of the most computationally expensive parts of the algorithm. Note that there may be a one-frame delay before the *activated* attribute is corrected due to this deferral if the child vertices are evaluated after the dependent vertex (line 2 of EVALUATE-VERTEX and lines 4–5 of NOTIFY). The function $branch_T(v)$ refers to the field of the parent in tree T that reflects the *enabled* field 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. boundary vertices).

5.1 Mesh Rendering

Once the vertex selection is made, a triangle mesh must be formed that connects the selected vertices. This mesh is defined by specifying the vertices encountered in a pre-order descent of the binary vertex trees. The recursive stopping condition is a **false** *enabled* attribute. To efficiently render the mesh, a triangle mesh graphics primitive, such as the one supported by IRIS GL and OpenGL [11, 15], 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 `glVertex()` call (OpenGL). A copy of the two-entry graphics vertex buffer, *my-buffer*, is maintained explicitly to allow the decision as to when to swap the entries to be made. The most recent vertex in this buffer is indexed by *ptr*.

The following pseudocode describes the mesh rendering algorithm. Each of the four triangular quadrants q_i are rendered in counterclockwise order, with the first vertex in each quadrant coincident with the last vertex in the previous quadrant (see Figure 2d). Hence, a single triangle mesh can be used to render the entire block. The indices q_{il}, q_{it}, and q_{ir} correspond to the left, top, and right vertex indices of quadrant q_i, respectively, with the "top" index being the center of the block. The block dimensions are 2^n+1 squared.

```
RENDER-BLOCK(b)
 1  enter triangle mesh mode
 2  render vertex v_q0l
 3  my-buffer_ptr ← q_0l
 4  previous-level ← 0
 5  for each quadrant q_i in block b
 6      if previous-level is even
 7          toggle ptr
 8      else
 9          swap vertices in graphics buffer
10      render vertex v_qil
11      my-buffer_ptr ← q_il
12      previous-level ← 2n+1
13      RENDER-QUADRANT(q_il, q_it, q_ir, 2n)
14  render vertex v_q0l
15  exit triangle mesh mode

RENDER-QUADRANT(i_l, i_t, i_r, level)
 1  if level > 0
 2      if enabled(v_it)
 3          RENDER-QUADRANT(i_l, (i_l+i_r)/2, i_t, level−1)
 4          if i_t ∉ my-buffer
 5              if level + previous-level is odd
 6                  toggle ptr
 7              else
 8                  swap vertices in graphics buffer
 9              render vertex v_it
10              my-buffer_ptr ← i_t
11              previous-level ← level
12          RENDER-QUADRANT(i_t, (i_l+i_r)/2, i_r, level−1)
```

The index $\frac{i_l+i_r}{2}$ corresponds to the (base) vertex that in the x-y plane is the midpoint of the edge $\overline{v_{i_l} v_{i_r}}$. Since *my-buffer* reflects what vertices are currently in the graphics buffer, line 9 in RENDER-BLOCK and line 8 in RENDER-QUADRANT could be implemented with a `glVertex()` call, passing the second most recent vertex in *my-buffer*.

6 DATA STRUCTURES

Many of the issues related to the data structures used with this algorithm have purposely been left open, as different needs may demand totally different approaches to their representations. In one implementation—the one presented here—as few as six bytes per vertex were used, and as many as 28 bytes per vertex were needed in another. In this section, we describe data structures that will be useful in many implementations.

For a compact representation, the vertex elevation is discretized and stored as a 16-bit integer. A minimum of six additional bits per vertex are required for the various flags, including the *enabled*, *activated*, and four *dependency* attributes. Optionally, the *locked* attribute can be added to these flags. The theoretical range of delta values becomes $[0, 2^{16}-1]$ in steps of $\frac{1}{2}$. We

elect to store each δ in "compressed" form as an 8-bit integer $\hat{\delta}$ in order to conserve space by encapsulating the vertex structure in a 32-bit aligned word. We define the decompression function as $\delta = \frac{1}{2}\lfloor(1+\hat{\delta})^{1+\hat{\delta}^2/(2^8-1)^2} - 1\rfloor$.[4] This exponential mapping preserves the accuracy needed for the more frequent small deltas, while allowing large delta values to be represented, albeit with less accuracy. The compression function is defined as the inverse of the decompression function. Both functions are implemented as lookup tables.

To accommodate tasks such as rendering and surface following, the vertices must be organized spatially for fast indexing. In Section 4.2, however, we implied that vertices within ranges of delta values could be immediately accessed. This is accomplished by creating an auxiliary array of indices, in which the entries are sorted on the corresponding vertices' delta values. Each entry uniquely references the corresponding vertex (i, j) via an index into the array of vertex structures. For each possible compressed delta value within a block, there is a pointer (index) $p_{\hat{\delta}}$ to a bin that contains the vertex indices corresponding to that delta value. The 2^8 bins are stored in ascending order in a contiguous, one-dimensional array. The entries in bin i are then indexed by $p_i, p_i+1, \ldots, p_{i+1}-1$ ($p_i = p_{i+1}$ implies that bin i is empty). For block dimensions up to 2^7+1, the indices can be represented with 16 bits to save space, which in addition to the 32-bit structure described above, results in a total of six bytes storage per vertex.

7 RESULTS

To show the effectiveness of the polygon reduction and display algorithm, we here present the results of a quantitative analysis of the number of polygons and delta projections, frame rates, computation and rendering time, and errors in the approximated geometry. A set of color plates show the resulting wireframe triangulations and textured terrain surfaces at different stages of the simplification and for different choices of τ. Two height field datasets were used in generating images and collecting data: a 64 km^2 area digital elevation model of the Hunter-Liggett military base in California, sampled at 2×2 meter resolution, and 1 meter height (z) resolution (Color Plates 1a–c and 2a–c); and a 1×1 meter resolution, 14 km^2 area of 29 Palms, California, with a z resolution of one tenth of a meter (Color Plates 3a–d). The vertical field of view is 60° in all images, which 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 Hunter-Liggett terrain was chosen for this purpose, which includes a variety of features such as ridges, valleys, bumps, and relatively flat areas. Figure 6 shows four curves drawn on a logarithmic scale (vertical axis). The top horizontal line, $n_0(\tau) = 13 \cdot 10^6$, shows the total number of polygons in the view frustum before any reduction method is applied. The curve second from the top, $n_1(\tau)$, represents the number of polygons remaining after the block-based level of detail selection is done. The number of polygons rendered, $n_2(\tau)$, i.e. the remaining polygons after the vertex-based simplification, is shown by the lowest solid curve. As expected, these two curves flatten out as τ is increased. The ratio $n_0(\tau)/n_2(\tau)$ ranges from about 2 ($\tau = 0$) to over 6,000 ($\tau = 8$). Of course, at $\tau = 0$, only coplanar triangles are fused. The ratio $n_1(\tau)/n_2(\tau)$ varies between 1.85 and 160 over the same interval, which clearly demonstrates the advantage of refining each uniform level of detail block.

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 the resulting surfaces, when textured, are seemingly identical to the ones obtained with no mesh simplification. Color Plates 1a–c illustrate the three stages 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, $n_0(1)/n_2(1)$ is slightly above 200, while $n_1(1)/n_2(1)$ is 18. The bottommost, dashed curve in Figure 6 represents the total number of delta values that fall in the uncertainty interval per frame (Section 4.2). 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, as out of the eight million vertices contained in the view frustum, only 14,000 evaluations of Equation 2 need to be made when $\tau = 1$.

[4] This results in an upper bound $\frac{2^{16}-1}{2}$ for the delta values.

τ	displacement				
	mean	median	max	std. dev.	$> \tau$ (%)
0.000	0.00	0.00	0.00	0.00	0.00
0.125	0.03	0.00	0.52	0.05	6.41
0.250	0.06	0.00	0.85	0.09	4.52
0.500	0.11	0.04	1.56	0.15	3.14
1.000	0.21	0.07	2.88	0.29	2.61
2.000	0.42	0.13	5.37	0.59	2.84
4.000	0.88	0.23	10.41	1.24	3.27
8.000	1.38	0.19	16.69	2.08	1.38

Table 1: Screen-space error in simplified geometry.

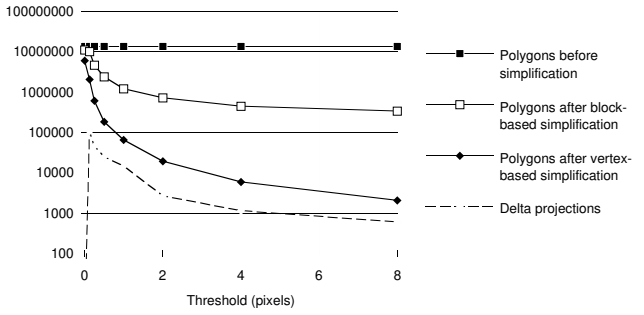


Figure 6: The number of polygons (n_0, n_1, n_2, from top to bottom) as a function of τ. The bottom curve shows the number of times Equation 2 was evaluated per frame.

In order to evaluate the errors due to the simplification, the points on the polygonal surface of the simplified mesh that have been displaced vertically, as well as the remaining triangle vertices, are perspective projected to screen-space and compared to the projections of the original set of vertices. Optimally, each such screen coordinate displacement should fall within the threshold distance τ. However, this constraint may in certain cases be violated due to the approximations discussed in Section 4.1. Table 1 was compiled for each mesh after vertex-based simplification was applied, and the surface points were correlated with the original eight million vertices shown in Color Plate 1a. The table summarizes the mean, median, maximum, and standard deviation of the displacements in number of pixels, as well as the fraction of displacements that exceed τ. In all cases, the average pixel error is well below τ. It can be seen that the approximations presented in Section 4.1 do not significantly impact the accuracy, as the fraction of displacements that exceed τ is typically less than five percent.

Color Plates 2a–c illustrate a checkerboard pattern draped over the polygonal meshes from Color Plates 1a–c. Qualitatively, these images suggest little or no perceivable loss in image quality for a

threshold of one pixel, even when the surface complexity is reduced by a factor of 200.

Figure 7 demonstrates the efficiency of the algorithm. The computation time associated with the delta projections (lines 10–20 in MAIN, Section 5) is typically only a small fraction of the rendering time. This data was gathered for the views shown in Color Plates 3a–d.

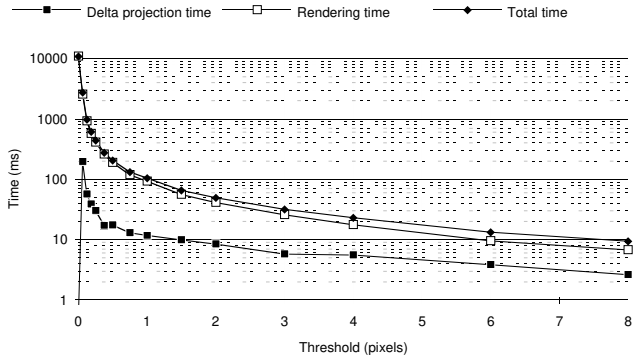


Figure 7: Rendering and evaluation times and their sum as functions of τ.

Figure 8 shows how the quantities in Figure 6, 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 over the Hunter-Liggett dataset. 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, but rather 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.

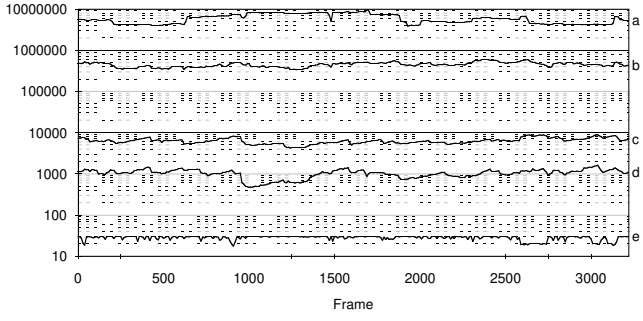


Figure 8: Time graph of (a) total number of polygons in view frustum, (b) number of polygons after block-based simplification, (c) number of polygons after vertex-based simplification, (d) number of delta projections, and (e) frames per second.

8 CONCLUSION

We have presented a height-field display algorithm based on real-time, per vertex level of detail evaluation, that achieves interactive and consistent frame rates exceeding twenty frames per second, with only a minor loss in image quality. Attractive features attributed to regular grid surface representations, such as fast geometric queries, compact representation, and fast mesh rendering are retained. 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 easily be changed interactively, with no incurred cost, determines the resulting image quality, and a direct relationship between this parameter and the number of rendered polygons exists, providing capabilities for maintaining consistent frame rates. The algorithm can easily be extended to handle the problem of gaps between blocks of different levels of detail, as well as temporal geometry morphing to further minimize popping effects.

Acknowledgement

This work was performed in part under contract DAKF11–91–D–0004–0034 from the U.S. Army Research Laboratory.

References

[1] AKELEY, K. RealityEngine Graphics. Proceedings of SIGGRAPH 93. In *Computer Graphics* Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 109–116.

[2] COSMAN, M. A., MATHISEN, A. E., and ROBINSON, J. A. A New Visual System to Support Advanced Requirements. In *Proceedings, IMAGE V Conference*, June 1990, pp. 370–380.

[3] DE BERG, M. and DOBRINDT, K. T. G. On Levels of Detail in Terrains. In *11th ACM Symposium on Computational Geometry*, June 1995.

[4] DE FLORIANI, L. and PUPPO, E. Hierarchical Triangulation for Multiresolution Surface Description. *ACM Transactions on Graphics* 14(4), October 1995, pp. 363–411.

[5] DOUGLAS, D. H. Experiments to Locate Ridges and Channels to Create a New Type of Digital Elevation Model. *Cartographica* 23(4), 1986, pp. 29–61.

[6] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., and STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. Proceedings of SIGGRAPH 95. In *Computer Graphics* Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH, pp. 173–182.

[7] FALBY, J. S., ZYDA, M. J., PRATT, D. R., and MACKEY, R. L. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers & Graphics* 17(1), 1993, pp. 65–69.

[8] FERGUSON, R. L., ECONOMY, R., KELLY, W. A., and RAMOS, P. P. Continuous Terrain Level of Detail for Visual Simulation. In *Proceedings, IMAGE V Conference*, June 1990, pp. 144–151.

[9] FOWLER, R. J. and LITTLE, J. J. Automatic Extraction of Irregular Network Digital Terrain Models. Proceedings of SIGGRAPH 79. In *Computer Graphics* 13(2) (August 1979), pp. 199–207.

[10] GARLAND, M. and HECKBERT, P. S. 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, 1991.

[12] GROSS, M. H., GATTI, R., and STAADT, O. Fast Multiresolution Surface Meshing. In *Proceedings of Visualization '95*, October 1995, pp. 135–142.

[13] HECKBERT, P. S. and GARLAND, M. Multiresolution Modeling for Fast Rendering. In *Proceedings of Graphics Interface '94*, 1994, pp. 1–8.

[14] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., and STUETZLE, W. Mesh Optimization. Proceedings of SIGGRAPH 93. In *Computer Graphics* Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 19–26.

[15] NEIDER, J., DAVIS, T., and WOO, M. *OpenGL Programming Guide*. Addison-Wesley, 1993.

[16] ROHLF, J. and HELMAN, J. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. Proceedings of SIGGRAPH 94. In *Computer Graphics* Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 381–394.

[17] SAMET, H. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys* 16(2), June 1984, pp. 187–260.

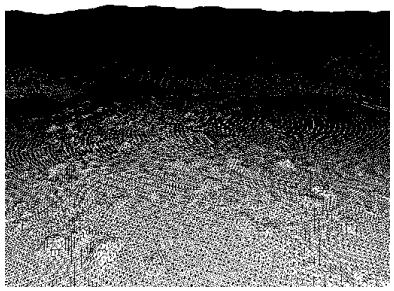
[18] SCARLATOS, L. L. A Refined Triangulation Hierarchy for Multiple Levels of Terrain Detail. In *Proceedings, IMAGE V Conference*, June 1990, pp. 114–122.

[19] SCHRODER, F. and ROSSBACH, P. Managing the Complexity of Digital Terrain Models. *Computers & Graphics* 18(6), 1994, pp. 775–783.

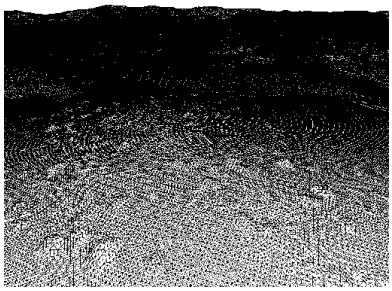
[20] SCHROEDER, W. J., ZARGE, J. A., and LORENSON, W. E. Decimation of Triangle Meshes. Proceedings of SIGGRAPH 92. In *Computer Graphics* 26(2) (July 1992), pp. 65–70.

[21] SOUTHARD, D. A. Piecewise Planar Surface Models from Sampled Data. *Scientific Visualization of Physical Phenomena*, June 1991, pp. 667–680.

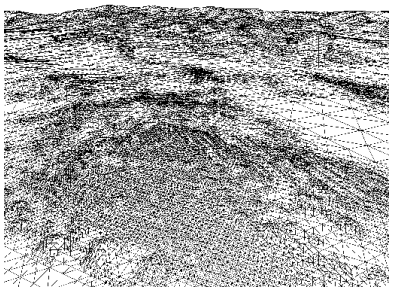
[22] TAYLOR, D. C. and BARRET, W. A. An Algorithm for Continuous Resolution Polygonalizations of a Discrete Surface. In *Proceedings of Graphics Interface '94*, 1994, pp. 33–42.



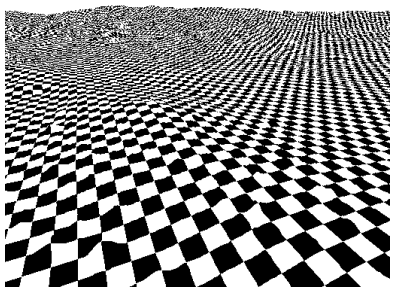
1a.



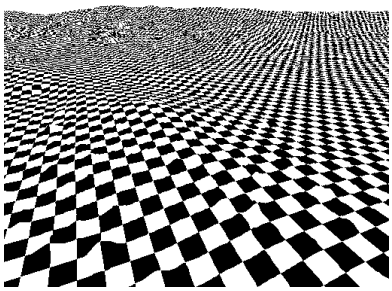
1b.



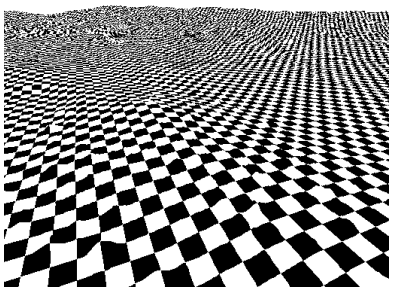
1c.



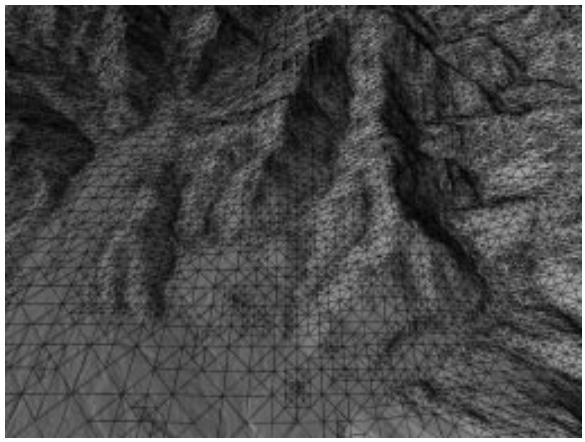
2a. Before simplification
13,304,214 polygons



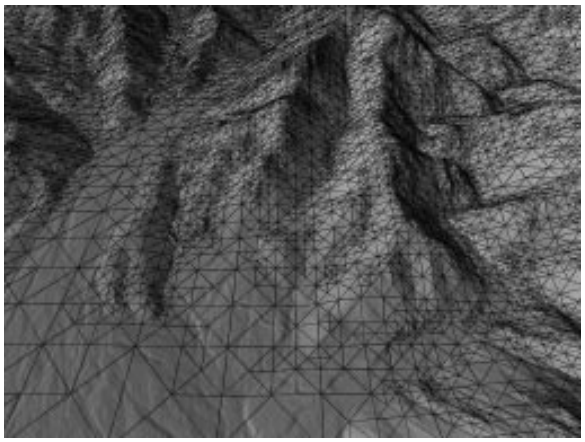
2b. After block−based LOD
1,179,690 polygons



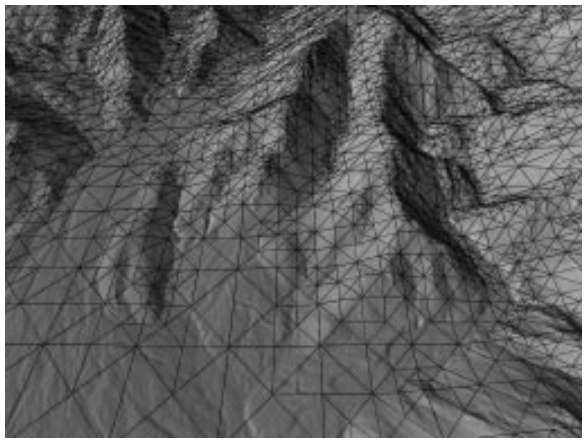
2c. After vertex−based LOD
64,065 polygons



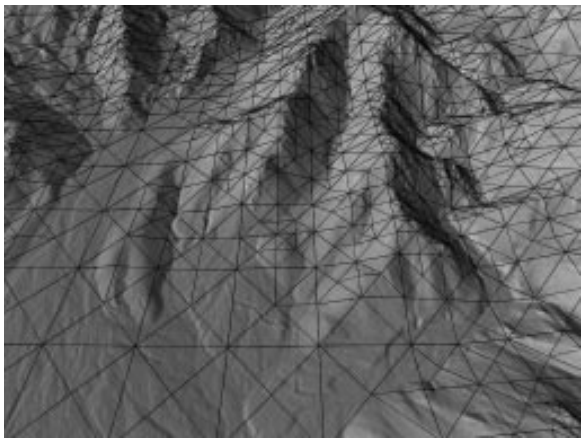
3a. $\tau = 0.5$, 62,497 polygons



3b. $\tau = 1.0$, 23,287 polygons



3c. $\tau = 2.0$, 8,612 polygons



3d. $\tau = 4.0$, 3,385 polygons