

# LEVEL-OF-DETAIL MANAGEMENT FOR REAL-TIME RENDERING OF PHOTOTEXTURED TERRAIN

Peter Lindstrom<sup>α</sup>, David Koller<sup>α</sup>, Larry F. Hodges<sup>α</sup>, William Ribarsky<sup>β</sup>,  
Nick Faust<sup>γ</sup> and Gregory Turner<sup>δ</sup>

<sup>α</sup> College of Computing and Graphics, Visualization & Usability Center  
Georgia Institute of Technology

<sup>β</sup> Office of Information Technology and Graphics, Visualization & Usability Center  
Georgia Institute of Technology

<sup>γ</sup> Center for GIS and Spatial Analysis Technologies and Georgia Tech Research Institute  
Georgia Institute of Technology

<sup>δ</sup> Information Processing Branch  
Army Research Lab

**Category:** Research Paper  
**Format:** Regular Paper

**Contact Author:** Larry F. Hodges, Ph.D.  
College of Computing  
801 Atlanta Drive, NW  
Georgia Institute of Technology  
Atlanta, GA 30058-0280

Phone: 404.894.8787  
FAX: 404.853.0673  
e-mail: hodges@cc.gatech.edu

## *ABSTRACT*

We present four techniques for real-time level-of-detail reduction of digital terrain data without loss of visual image quality or detail. Techniques for reducing polygon count based on distance and terrain roughness provided two-orders-of-magnitude reduction in the number of polygons rendered for our sample data: an eight kilometer by eight kilometer data set with four meter resolution. Techniques for reducing texture data based on distance and orientation of polygons resulted in a one-order-of-magnitude reduction in the number of bytes of texture memory for the same data set.

# LEVEL-OF-DETAIL MANAGEMENT FOR REAL-TIME RENDERING OF PHOTOTEXTURED TERRAIN

**Category: Research paper**  
**Format: Regular paper**

## *ABSTRACT*

We present four techniques for real-time level-of-detail reduction of digital terrain data without loss of visual image quality or detail. Techniques for reducing polygon count based on distance and terrain roughness provided two-orders-of-magnitude reduction in the number of polygons rendered for our sample data: an eight kilometer by eight kilometer data set with four meter resolution. Techniques for reducing texture data based on distance and orientation of polygons resulted in a one-order-of-magnitude reduction in the number of bytes of texture memory for the same data set.

## 1. BACKGROUND

Real-time visual simulation systems require transformation of scene data in real-time from some persistent storage format to renderable scene object format. The data in these systems is typically heterogeneous (composed of multiple classes) and massive in size (up to terabytes). Systems of this type are crucial to the development of current experimental prototypes for real-time battlefield visualization in DoD, to Distributed Interactive Simulation (DIS) Systems (Macedonia and Zyda, 1994), and to further development of the area of Geographic Information Systems (Faust, et. al., 1994; Brown and Pike, 1993). Advances in visualization, interactive techniques for analysis, parallel computing, and distributed computing offer the components for a highly integrated, efficient, real-time visual simulation systems with truly immersive capability for navigating and understanding complex, constantly changing information databases. In our current research, Georgia Tech and the Information Processing Branch of the Army Research Lab (ARL) have developed critical techniques for bringing these components together into a working system. As part of this system we have developed a suite of real-time level-of-detail analysis techniques that implement a dynamic balancing between frame-rate and terrain model resolution in order to achieve realistic

interaction (such as head-motion parallax in virtual reality systems) while maintaining visual fidelity.

Our techniques address a subset of the general problem of multi-resolution modeling for fast rendering (Clark, 1976; Heckbert and Garland, 1994). Digital terrain models primarily deal with meshes of elevation values and phototextures that map to those meshes. Previous work has concentrated on algorithms to construct simplified polygonal models or to decide when to use those models. Approaches in the first category have included techniques such as constructing a triangular mesh that closely approximates a surface while using a small number of triangles (Scarlatos and Pavlidis, 1992), adaptive subdivision to fit a set of polygons to measured data points (DeHaemer and Zyda, 1991), or decimation algorithms to simply remove data points (Williams, 1983; Schroeder, Zarge and Lorensen, 1992)). Approaches in the second category include cost-benefit heuristics to render the best possible image consistent with maintaining a minimum frame rate (Funkhouser and Sequin, 1993) and rendering at different resolutions based on distance from the observer (Falby, et al., 1993).

Our work falls primarily into the second category. Using a pre-computed hierarchical set of data bases of different levels-of-detail

for elevation and texture data, we do real-time determination of the level-of-detail needed to maintain image fidelity. Our approach provides, on the average, a two-orders-of-magnitude reduction in polygons and a one-order-of-magnitude reduction in texture memory for digital terrain models while producing images that have the same visual quality as those computed at full resolution.

Our methods for adjusting levels of detail according to terrain distance, roughness, and texture resolution based on viewing angle could be part of a more general level-of-detail management model. Hitchner and McGreevy (1993) have developed such a general model that includes these factors in conjunction with hierarchical or other multiple model approaches. Such an analytic model, which allows for evaluation of system load functions, user interest weight functions, and criterion functions, provides a detailed method for meeting performance requirements. It is also quite useful because it offers a general framework for adapting, building upon, or quantitatively evaluating a management model. The research described here provides a detailed implementation and performance evaluation of significant factors for such a model.

## 2. TERRAIN REPRESENTATION

The external terrain model is represented by a uniform square grid of points, where each point has two attributes--elevation and color. The terrain surface is visualized as a textured polygonal mesh. For rendering purposes, we break the terrain surface up into triangles rather than polygons of an arbitrary number of sides.

In general, the number of polygons for a typical database is on the order of several million. When doing operations on these polygons (e.g. culling and selection of rendering detail--see sections below), we can gain speed by grouping them into blocks of polygons and then treating each block as an independent entity. For efficiency reasons, we use a *quad tree* as a hierarchical internal representation of the terrain model. The top

level node in this tree represents the area of the entire database, its children each represent one fourth of the terrain area, their children in turn each cover one sixteenth of the area, etc. Depending on the dimensions of the database, which are constrained to powers of two plus one<sup>1</sup>, we create a tree deep enough to split the terrain into blocks, or *quad cells*, of  $x$  and  $y$  dimensions of either 65 or 129 points<sup>2</sup>. We currently store one single contiguous array for the whole topographic database, while the texture map must be cut up into pieces that are then dynamically paged into a relatively small texture memory cache. The highest resolution texture blocks belong to the lowest level of the tree, and the texture resolution is successively reduced at every level up in the tree. This allows us to use a single texture block size for all resolutions. The following sections describe the algorithms used to reduce the amount of data required to accurately render a scene.

## 3. VIEW FRUSTUM CULLING

One advantage of using a quad tree representation is that it allows efficient polygon culling. In order to avoid excessive time spent rendering polygons that are not within the field-of-view, we intersect the terrain with the view frustum and render only those quad cells that are part of this intersection. For each leaf node in the quad tree, we define a bounding box that contains all of the polygons assigned to that cell. The bounding box of a parent cell is then defined as the smallest box that contains the bounding boxes of all its children. The culling is done in a recursive preorder manner--if a quad cell is not within the field-of-view as determined by the intersection with its bounding box, none of its children can be visible, and we move back up a level in the recursion. If, however, the quad cell is visible, its children are recursively checked against the view frustum. Thus, for non-visible quad cells in

---

<sup>1</sup>This condition is necessary in order to obtain equal size areas of all quad cells on the same level.

<sup>2</sup>The dimensions used depend on the total number of texture blocks resident in texture memory which is limited to 512 on our SGI Reality Engine.

the top levels of the tree, we can cut out large pieces of the terrain that do not need to be rendered.

While the relatively low granularity provided by the quad tree causes many polygons not in the field-of-view to be rendered, quad cell culling is a very inexpensive way of reducing the number of polygons considered for rendering. For a field-of-view of 90 degrees, the culling stage generally reduces the number of polygons to less than half. Furthermore, culling plays an important role in fitting texture into the limited amount of hardware texture memory available.

## 4 MULTIPLE LEVEL-OF-DETAIL GEOMETRY RENDERING

While view frustum culling of terrain data reduces the amount of data to be rendered, we are still left with a number of polygons that greatly exceeds the rendering capabilities of current hardware architectures when real-time animation is required. In order to cope with this large amount of data, we must simplify our model and render fewer but larger polygons wherever possible while still maintaining image quality. Since rendering speed is usually inversely proportional to image quality, we could specify a lowest acceptable rendering speed and obtain the highest image quality possible within that constraint (Funkhouser and Sequin, 1993). Since image quality is crucial to our driving application, we take a different approach and set a lower bound on acceptable image quality. The algorithms described in this paper choose an appropriate level-of-detail for rendering of each quad cell such that consistent image quality is maintained and rendering time is minimized subject to that constraint.

### 4.1 DISTANCE BASED POLYGON RESOLUTION

Perspective projection causes distant polygons to appear smaller on the screen than polygons close to the viewer. At some distance, the vertices that make up a polygon are all going to render into the same pixel on

the screen. At this point a high degree of terrain grid resolution becomes a potential liability instead of an asset. In the best case the pixel representation is a blended version of all the polygons that are rendered into that pixel, an effect that could have been achieved with less computation and a coarser data base. In the worst case the way the z-buffer handles depth values can cause both visibility and color inconsistencies to appear in the pixel.

Our algorithm determines the correct distance at which a smaller set of polygons may be used to approximate the terrain surface. Currently the reduced data set is created by decimation of every other grid elevation point, resulting in a factor of four reduction in the number of polygons that must be rendered. As the distance to the viewpoint increases, the same technique is applied repeatedly to reduce the data set further. In order to determine the minimum distance at which a lower polygon resolution would be appropriate, we first select what we call a *pixel threshold*. When the projected distance between the vertices of a polygon is smaller than this threshold (denoted  $t_{pd}$ ), we use the next lower polygon resolution. If these polygons in turn project smaller than the threshold, an even lower resolution is used until the sizes of the projected polygons exceed the threshold. The distances from the viewpoint to the boundaries between different polygon resolutions, or *resolution cutoffs*, are pre computed by extending two lines defined by the viewpoint and two points on the projection plane that are separated by the threshold. The first cutoff is found where the distance between these lines equals the highest vertex resolution (i.e. the spacing between the vertices in the terrain model). The second cutoff equals the distance to the point where the lines are separated by twice the vertex resolution, the third cutoff is at the point where the distance between the lines is four times the vertex resolution, and so on (see Figure 1). To guarantee maximum image quality, an optimal threshold value of one pixel should be used. However, our experience is that a threshold of up to four pixels can often be used without a significant loss of image quality.

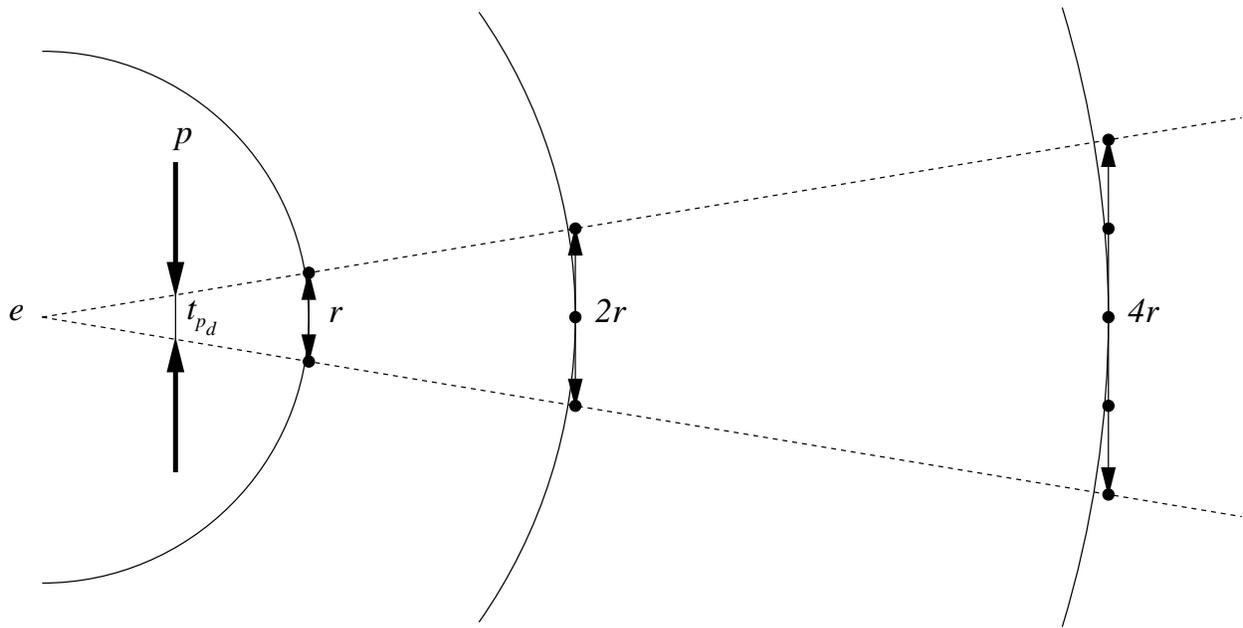
Since all culling and rendering is done on a per quad cell basis, each quad cell is assigned a uniform polygon resolution. To determine quad cell resolutions, we first surround each cell by a sphere whose diameter equals the length of the longest diagonal of the cell's bounding box. The distance to the center of a quad cell minus the radius of its bounding sphere is compared to each cutoff to determine the overall polygon resolution for that cell. This ensures that all the polygons of a quad cell are beyond a cutoff before the resolution associated with that cutoff is used to render the cell.

## 4.2 TERRAIN ROUGHNESS

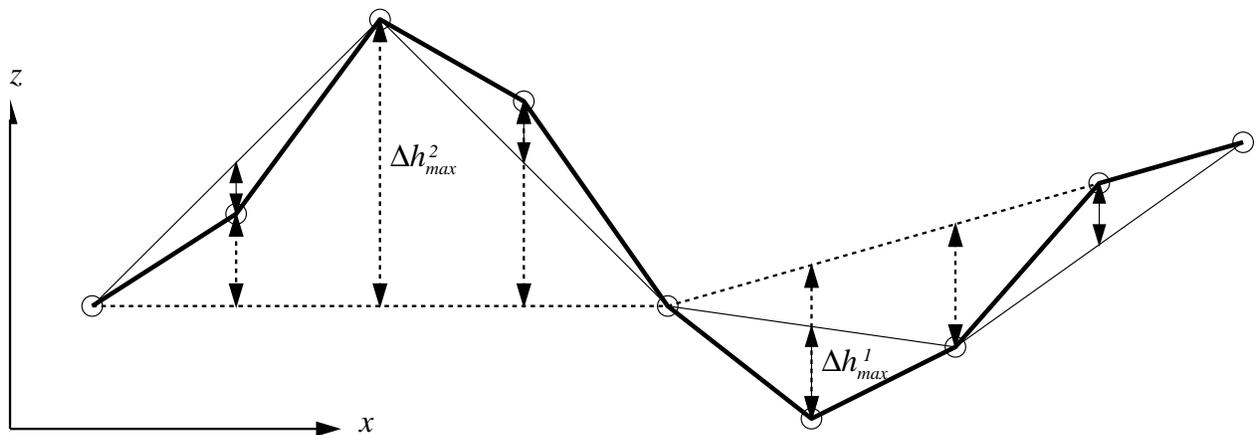
The second constraint on image quality is given by the roughness of each quad cell. Where the topography is fairly smooth and only minor differences in the change in elevation exist, a lower polygon resolution can be used. Consider the extreme case where all the vertices of a quad cell lie in the same plane. For this case, we can save a substantial amount of work by rendering only one polygon (or two triangles covering the quad cell area) since its surface exactly equals the surface produced by rendering each individual smaller polygon. Note that this can be done even if the vertex elevations are not constant as long as they all lie in one plane. Even if the vertices do not all lie in the same plane, small discrepancies between a high and a low resolution may not be detectable beyond a certain distance, and the lower resolution can be used.

In order to determine when the quad cell roughness is small enough to use a lower resolution, we look at the projected difference in elevation, or *elevation error*, resulting from using a lower than maximum resolution (since the terrain grid is uniform, lower resolution polygons do not impose changes in the  $x$  and  $y$  components). In general, there is a difference in elevation between the points on a low resolution polygon and the higher resolution polygons spanned by the low resolution polygon (e.g. a dent in the terrain may be filled by a larger lower resolution polygon covering the dent). By computing the elevation errors (denoted  $\Delta h$ ) of all polygons of a certain resolution, and projecting these errors onto the projection plane, we can decide whether using a lower resolution is acceptable by comparing the errors to a threshold (denoted  $t_{pr}$ ). Figure 2 shows a two-dimensional  $x$ - $z$  view of the terrain viewed from the side and the elevation errors for various resolutions.

It would be very expensive to compare all of the  $\Delta h$ 's for every polygon in each quad cell with the threshold for each frame, so we simplify the task by either taking the maximum ( $\Delta h_{max}$ ) or the mean ( $\Delta h_{mean}$ ) of all polygon elevation errors for each quad cell and doing only one projection/comparison per frame for each cell and resolution. ( $\Delta h_{max}$  or  $\Delta h_{mean}$  is assumed to be positioned at the center of the quad cell for the projection.) The maximum is used when image quality is of



**Figure 1. Resolution Cutoffs.**  $r$  represents the database vertex resolution, while  $t_{pd}$  is the pixel threshold.



**Figure 2. Elevation errors for various resolutions.**  $\Delta h_{max}^1$  and  $\Delta h_{max}^2$  are the local error maxima for resolutions 1 and 2 respectively.

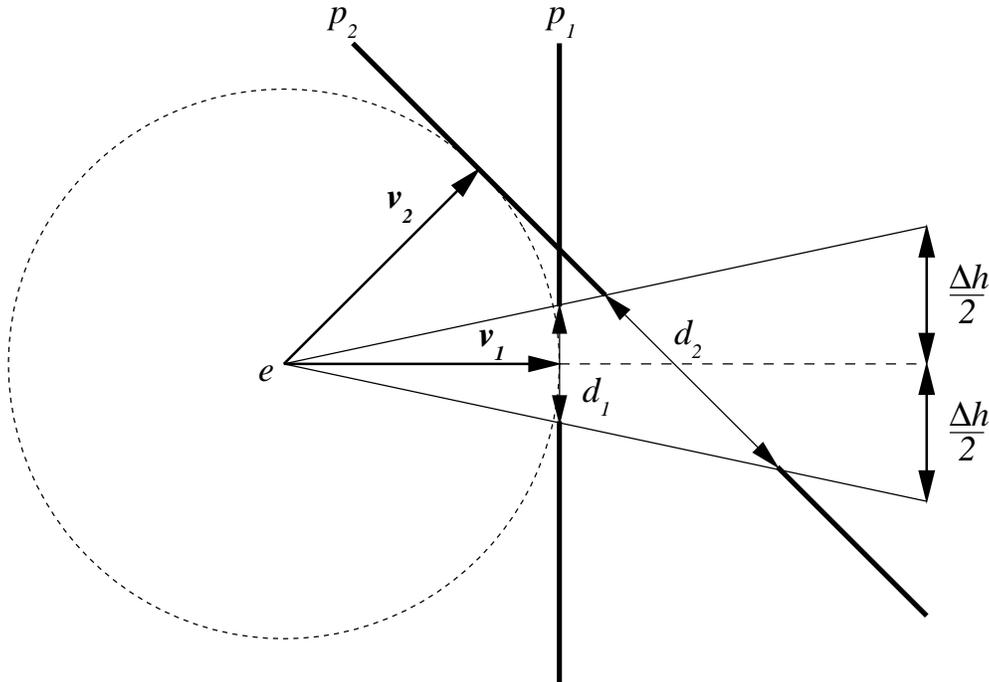
great importance, while the mean accompanied by a possibly smaller threshold can be used to decrease the influence on resolution caused by spikes and other outliers in the terrain data.

By using perspective projection of the height errors for the roughness test, we introduce an undesired effect--the quad cell resolution increases as the projected cell moves from the center of view toward the periphery of the screen. This is illustrated in Figure 3 where  $d_1$ , the projection of  $\Delta h$  onto plane  $p_1$  when  $\Delta h$  is perpendicular to the line of sight, is smaller than  $d_2$ , the projection onto plane  $p_2$  of  $\Delta h$  when viewed at an angle from the same viewpoint ( $e$  in Figure 3). As the viewing direction changes, a cell in the center of the screen is rendered in a resolution that is relatively lower than the resolution used when the same cell is rendered close to the edges of the screen. To compensate for this effect when testing for roughness, we assume that the projection plane is always perpendicular

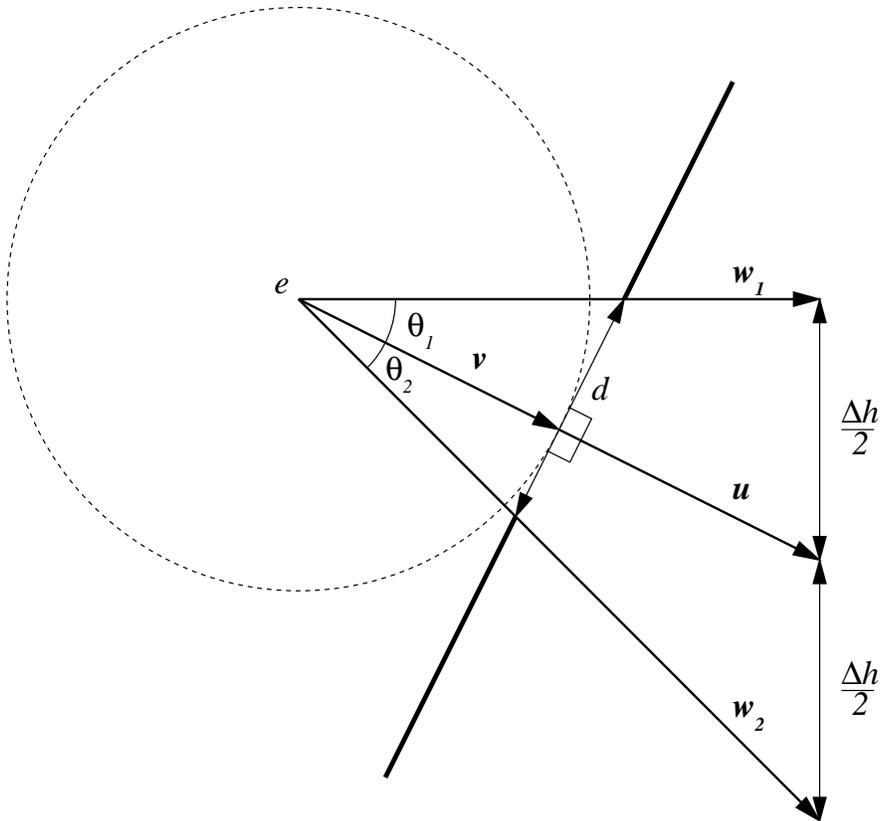
to the line connecting the viewpoint and the center point of each quad cell (see Figure 4 ). The projection of  $\Delta h$  then becomes

$$\begin{aligned} d &= v(\tan(\theta_1) + \tan(\theta_2)) \\ &= v\left(\frac{\sin(\theta_1)}{\cos(\theta_1)} + \frac{\sin(\theta_2)}{\cos(\theta_2)}\right) \\ &= v\left(\frac{\left\|\mathbf{u} \times \left(\mathbf{u} + \frac{\mathbf{h}}{2}\right)\right\|}{\mathbf{u} \cdot \left(\mathbf{u} + \frac{\mathbf{h}}{2}\right)} + \frac{\left\|\mathbf{u} \times \left(\mathbf{u} - \frac{\mathbf{h}}{2}\right)\right\|}{\mathbf{u} \cdot \left(\mathbf{u} - \frac{\mathbf{h}}{2}\right)}\right) \\ &= v\left(\frac{\|\mathbf{u} \times \mathbf{w}_1\|}{\mathbf{u} \cdot \mathbf{w}_1} + \frac{\|\mathbf{u} \times \mathbf{w}_2\|}{\mathbf{u} \cdot \mathbf{w}_2}\right) \end{aligned}$$

Here the vector  $\mathbf{h}$  has magnitude  $\Delta h$  and points in the positive  $z$  (up) direction.  $v$  is the magnitude of vector  $\mathbf{v}$ .



**Figure 3. Projections of  $\Delta h$  onto different planes for the same viewpoint  $e$ .**



**Figure 4. Projection of the elevation error  $\Delta h$ .**  $\mathbf{u}$  is the vector from the viewpoint,  $e$ , to the center of the quad cell.

The roughness comparison takes not only the viewing angle into consideration--a top-down view yields small  $\Delta h$  projections and a low resolution can be used to render the scene -- but a distance comparison is also inherent in this algorithm. The reason we still maintain a separate distance comparison is because for certain terrain databases, we may want to completely turn off the more expensive roughness computations and base resolution entirely on distance. When both constraints are turned on, we use the lowest resolution that meets both of the threshold criteria. In general, the threshold associated with the distance from the viewpoint  $t_{pd}$  is used only as an upper bound on the on-screen polygon size.

## 5 MULTIPLE LEVEL OF DETAIL TEXTURE RENDERING

Similar to the polygon resolution which determines the shape of the rendered geometry, we use multiple texture resolutions which maintain the appearance and detail of the color components of each polygon. When determining the texture resolutions, we employ a strategy similar to the process used in selecting polygon resolutions. The following two sections discuss the similarities and differences in texture versus polygon resolution.

### 5.1 DISTANCE BASED TEXTURE RESOLUTION

As for the distance based polygon resolution described in Section 4.1, we define a series of cutoffs for texture resolution. These cutoffs are, however, based on a separate threshold,

$t_{td}$ , which is usually made smaller than  $t_{pd}$  since the image quality is affected to a greater extent by color cues than the spatial cues associated with the terrain geometry. When this threshold is significantly larger than one pixel, the scene appears blurry as each screen pixel does not map to a unique texture pixel (or *texel*). When the spacing between polygon vertices at the highest existing resolution for a given terrain model are separated by more than one pixel, we do color interpolation between the pixels<sup>3</sup> to avoid a blocky and jagged appearance of terrain texture.

To obtain multiple resolution textures, we pre interpolate the highest resolution texture blocks. Each resolution texture block covers the same area as four texture blocks of the next higher resolution. Texture block dimensions are constant, so each texel corresponds to the interpolated colors of four texels in the next higher resolution texture blocks.

## 5.2 TEXTURE RESOLUTION BASED ON THE VIEWING ANGLE

To further decrease the amount of texture data needed to render a scene, we take advantage of the fact that as a polygon is rotated away from the viewer, it occupies less area (number of pixels) on the screen than if it is perpendicular to the line of sight. A polygon of length  $l$  projects to a screen length  $d$  (see figure 5). As  $d$  decreases with polygon rotation, lower texture resolutions may be used to render the polygon. Thus, when polygons are viewed from the side, a lower texture resolution can be used. This relationship may be specified by:

$$\begin{aligned} d &= d_1 + d_2 \\ &= v \left( \frac{b}{u+a} + \frac{b}{u-a} \right) \\ &= v \frac{l}{2} \cos(\varphi) \left( \frac{1}{u + \frac{l}{2} \sin(\varphi)} + \frac{1}{u - \frac{l}{2} \sin(\varphi)} \right) \\ &= vl \cos(\varphi) \left( \frac{1}{2u + l \sin(\varphi)} + \frac{1}{2u - l \sin(\varphi)} \right) \\ &= vl \cos(\varphi) \frac{4u}{4u^2 - l^2 \sin^2(\varphi)} \end{aligned}$$

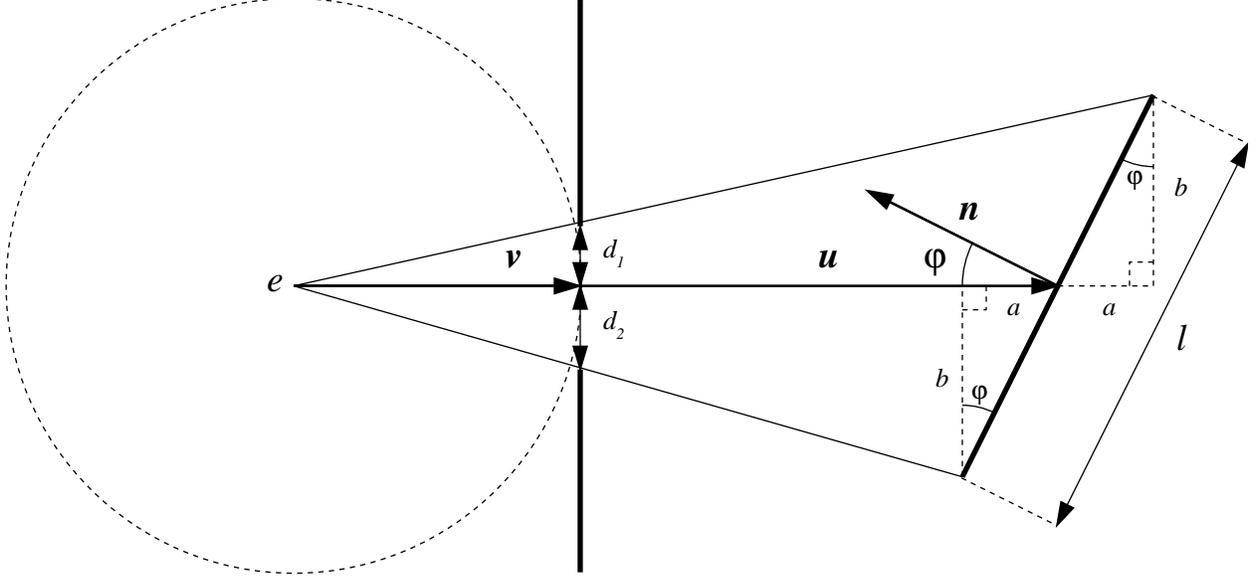
In general  $u \gg l$ , so we can approximate by

$$d \approx \frac{vl}{u} \cos(\varphi)$$

which is the same as the projection of the polygon when it is perpendicular to  $\mathbf{u}$  times the cosine of the angle created by  $-\mathbf{u}$  and the polygon normal  $\mathbf{n}$ . As in Section 4.2, we assume that the projection plane is perpendicular to the vector  $\mathbf{u}$  defined by the viewpoint and the center of the polygon.

For a given quad cell and viewpoint, we can find the maximum  $\cos(\varphi)$  for all polygons in the cell and adjust the texture resolution according to this value. This adjustment is done most easily by multiplying each cutoff by  $\cos(\varphi)_{max}$  since the perspective projection is inversely proportional to the viewpoint distance. Following the same reasoning as in previous sections, we would like to do computations on a per quad cell basis rather than on a per polygon basis because of the very large number of polygons. Then how do we find  $\cos(\varphi)_{max}$ , or the polygon normal closest to the vector  $-\mathbf{u}$  for a given view? Rather than finding one single normal, we approach this problem by defining a volume that contains all of the normals for a given quad cell.

<sup>3</sup>This is automatically done by SGI GL when texture filters are set to TX\_BILINEAR.



**Figure 5.** The projection of a polygon edge of length  $l$ . The vector  $\mathbf{n}$  is the polygon normal.

If we assume that all of the normals originate at the center of the quad cell, we can define a circular cone that encloses all of these normals (a justification of this assumption will be given below). If the vector  $\mathbf{w}$  from the center of the quad cell to the viewpoint is contained in this cone, we assume that there exists a normal that coincides with  $\mathbf{w}$  and  $\cos(\varphi)_{max} = 1$ . Otherwise, we choose  $\cos(\varphi)_{max}$  as the cosine of the smallest angle between  $\mathbf{w}$  and the vectors defining the surface of the cone (see Figure 6).

In order to compute the cone enclosing the polygon normals for a given quad cell, we first scale the normals to unit length. The normals are then projected onto the  $x$ - $y$  plane by discarding the  $z$  components. We now find the maximum magnitude  $r_{max}$  of the projected normals and form the cone by the center point  $\mathbf{c}$  of the quad cell and the circle

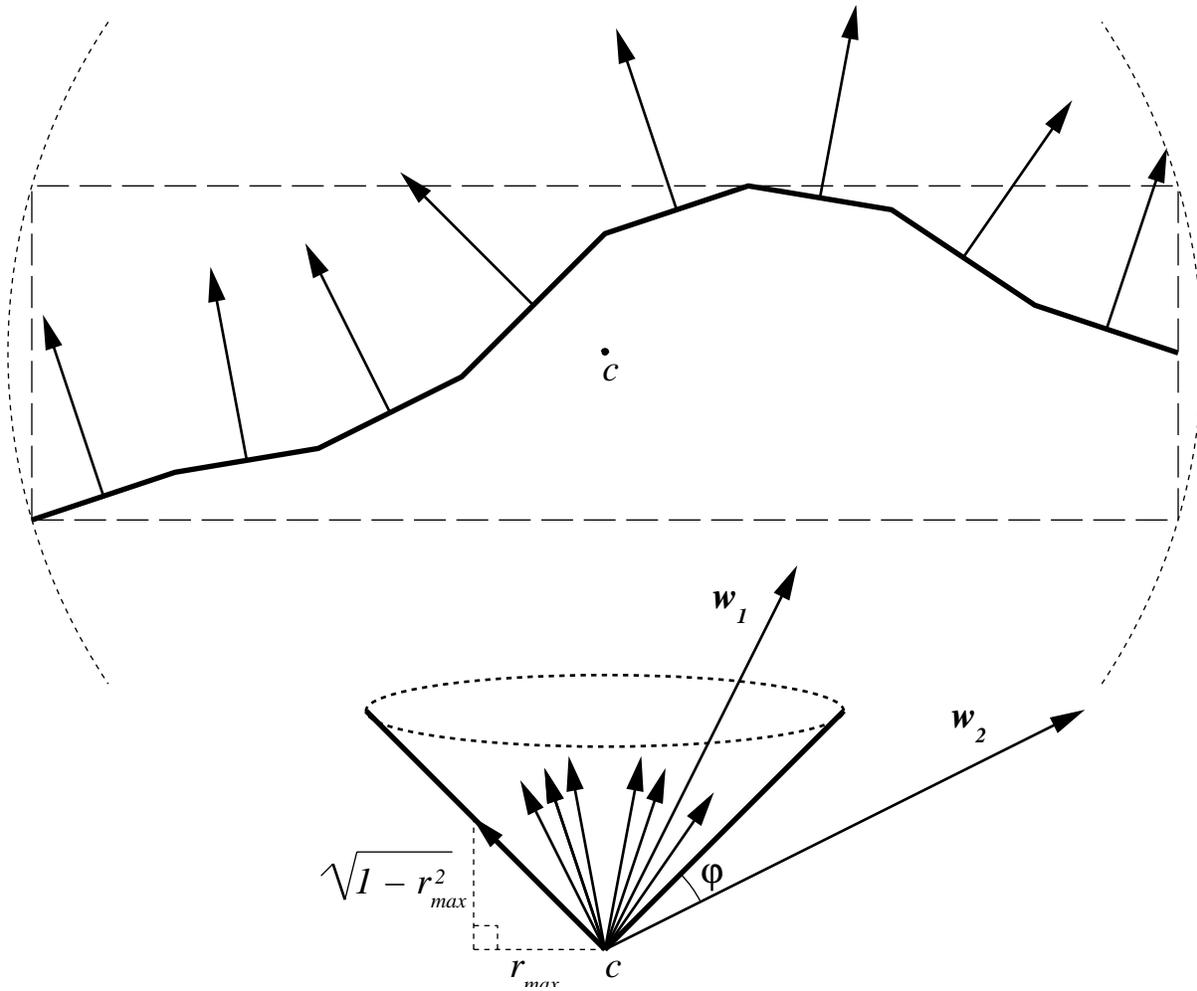
$$\left[ c_x + r_{max} \cos(\theta), c_y + r_{max} \sin(\theta), c_z + \sqrt{1 - r_{max}^2} \right]$$

for  $0 \leq \theta \leq 2\pi$ .  $\cos(\varphi)_{max}$  is then defined by

$$\begin{aligned} \cos(\varphi)_{max} &= w_x \frac{r_{max} w_x}{\sqrt{w_x^2 + w_y^2}} \\ &+ w_y \frac{r_{max} w_y}{\sqrt{w_x^2 + w_y^2}} + w_z \sqrt{1 - r_{max}^2} \\ &= \sqrt{w_x^2 + w_y^2} r_{max} + w_z \sqrt{1 - r_{max}^2} \end{aligned}$$

assuming  $\mathbf{w}$  is outside of the cone and has been normalized to unit length.

Spikes in the terrain data may cause the cone to be very wide, but we can often get away with using the average radius  $r_{mean}$  rather than the maximum radius  $r_{max}$ . By moving all of the normals to the center of the quad cell, we lose some accuracy in the computation of  $\cos(\varphi)$ . However, when the distance to a quad cell is large, the relatively small displacement to the center of the quad cell has little influence on the vector  $\mathbf{w}$  (i.e.,  $\mathbf{w} \approx -\mathbf{u}$ ). When the viewpoint is very close to a quad cell, the subtraction of the radius of the sphere surrounding the quad cell (see Section 4.1) yields a negative number, so comparison with a decreased cutoff still results in use of the highest texture resolution.



**Figure 6. Bounding cone for polygon normals.**  $w_1$  is inside the cone, while  $w_2$  makes an angle  $\phi$  with the cone. This 2D view of a quad cell shows the cell's bounding box, bounding sphere, and center point  $c$ .

## 6. SOME RENDERING DETAILS

After culling has been done and polygon and texture resolutions have been determined, we render the scene. Because the *SGI Reality Engine* requires each texture to be defined before it can be used, and because the amount of texture memory is in general not sufficiently large to contain the entire database, we chop the terrain up into smaller texture blocks and dynamically bind each texture block at render time (Akeley, K, 1993). As mentioned in Section 2, each quad cell is given a texture block of a resolution determined by its position in the quad tree--

the highest resolution textures are assigned to leaf nodes, the next highest resolution blocks are found one level above, and so on. Rendering is done on a per quad cell basis, so it is the texture resolution of a given quad cell that determines on what level of the quad tree the cell will be rendered. We take advantage of SGI RE's triangle mesh primitive to obtain maximum rendering speed.

Theoretically, the total number of pixels on the screen should determine the amount of texture memory required to render a scene. In practice, however, a substantially larger amount of texture memory is required which

is due in part to the per quad cell, as opposed to per pixel, operations (e.g. some part of, or maybe even all of a rendered quad cell may not even appear on the screen even though culling is done). If the texture memory gets overloaded, SGI GL will automatically swap the texture block that last rendered onto the screen. If a swapped out texture block is needed for the next frame, we must--to the greatest extent possible--avoid trading this texture block for a block which is also used to render the scene. If we always render quad cells in the same order, we may end up chasing our own tail by successively swapping textures out right before we need them (compare this to the problems with *least recently used* paging in virtual memory systems). We solve this by reversing the order in which quad cells are rendered between successive frames. Thus, only a fraction of the texture memory needs to be swapped per frame.

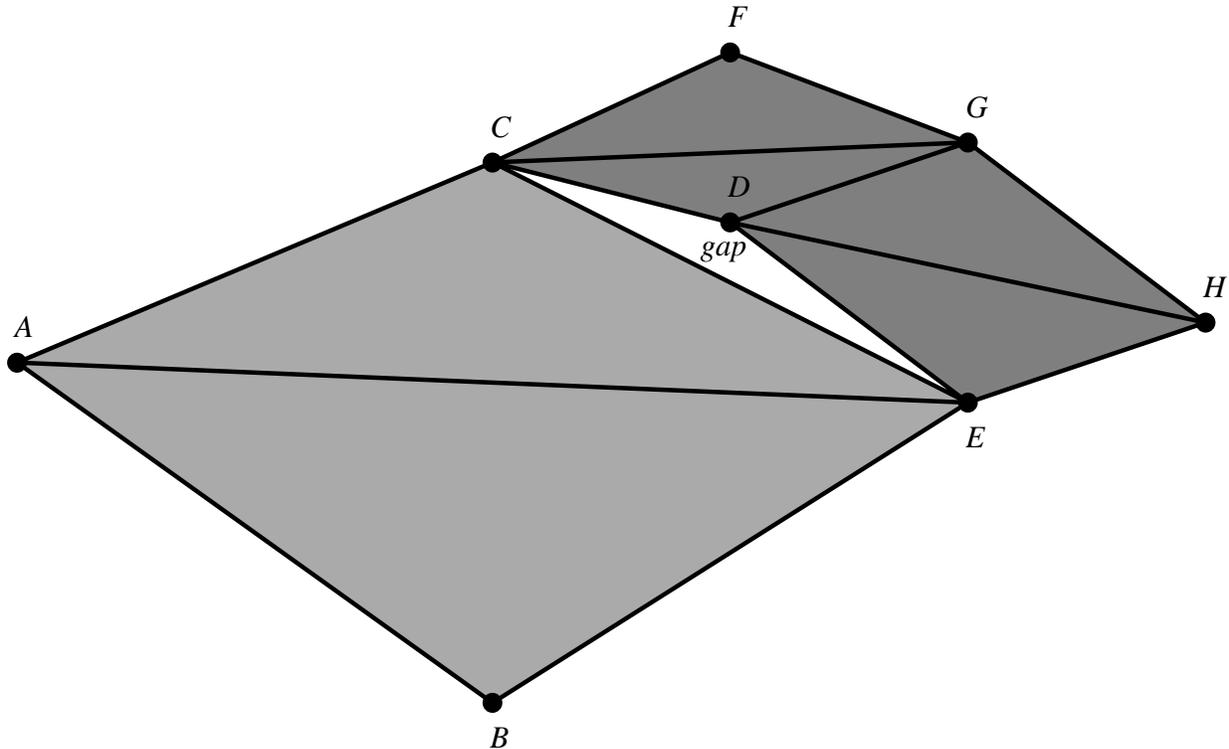
One of the problems associated with dynamic changes of polygon resolution is connecting cells of different resolutions. Most often, small gaps along the quad cell edges will appear since all of the points on an edge are not shared by two cells of different resolution (see Figure 7 where point *D* is not common to both resolutions). One solution to this problem is given in Dehaemer and Zyda (1991), where the vertices of the higher resolution along with the points of the lower resolution are all used to form many-sided polygons that cover the gaps (i.e., the points *A*, *B*, *C*, *D*, and *E* in Figure 7 would be used to form a pentagon). One drawback of this approach is the potential non-planarity and concavity inherent in such polygons, and rendering of such polygons is often implementation dependent with sometimes unpredictable results. Our current solution is somewhat more simplistic--we use a number

of triangles in the vertical  $x$ - $z$  and  $y$ - $z$  planes to fill in the gaps (in Figure 7 we would draw the triangle *CDE*). This approximation works well in textured terrains and has little impact on the image quality. For shaded images of the terrain where texture is not used, the transition between two cells of greatly differing resolutions is less smooth however. An algorithm is currently being implemented that guarantees smooth transitions and that only uses triangle meshes.

## 7. PERFORMANCE

The underlying philosophy of our approach to detail reduction is that there is a finite number of pixels making up the display area and that the modeled detail should match but not exceed the available screen resolution. As an example of the performance of our algorithms we present performance measures taken from a typical data set. Test area is eight kilometers by eight kilometers with elevation values on a regular grid at four meter intervals. Total number of polygons in the model is 8,388,608. Total number of bytes of texture is 12,582,912. Computed pixel resolution was 640 by 480.  $t_{pd} = 64$ ,  $t_{pr} = 0.5$ , and  $\Delta h_{mean}$  is used for polygon reduction.  $t_{td} = 1$  and the mean radius were used for texture reduction.

Figures 8 and 9 contain performance data taken from an animated view from the top of a tank moving across the terrain during a 445 second period. Data values were taken every 120 frames. Average frame rate was 17.5 frames/second (s.d. = 1.8 frames). Minimum frame rate for any 120-frame-sequence was 13.6 frames/second.



**Figure 7. Connecting cells of different resolution.**

Square data points in Figure 8 represent the total number of polygons in the view volume after view frustum culling (mean = 3,026,130, s.d. = 1,001,650). Triangular data points represent the number of polygons actually rendered after data reduction based on distance and terrain roughness (mean = 18,749, s.d. = 3,498). On the average the number of polygons rendered is reduced to less than 1% of the polygons in the view frustum. Also note that large changes in the number of polygons in the view volume are smoothed by the data reduction algorithms so that the graph of the number of rendered polygons is much smoother than the graph of the number of polygons in the view volume. Similar results are found in the texture data as shown in figure 9. Square data points represent the number of bytes of texture in the view frustum for each frame (mean = 4,339,194, s.d. = 1,502,474), while triangular data points represent the number of bytes of texture rendered (mean = 190,919, s.d. = 28,831). The number of bytes of texture rendered is reduced to less than 5% of the number of bytes of texture in the view frustum.

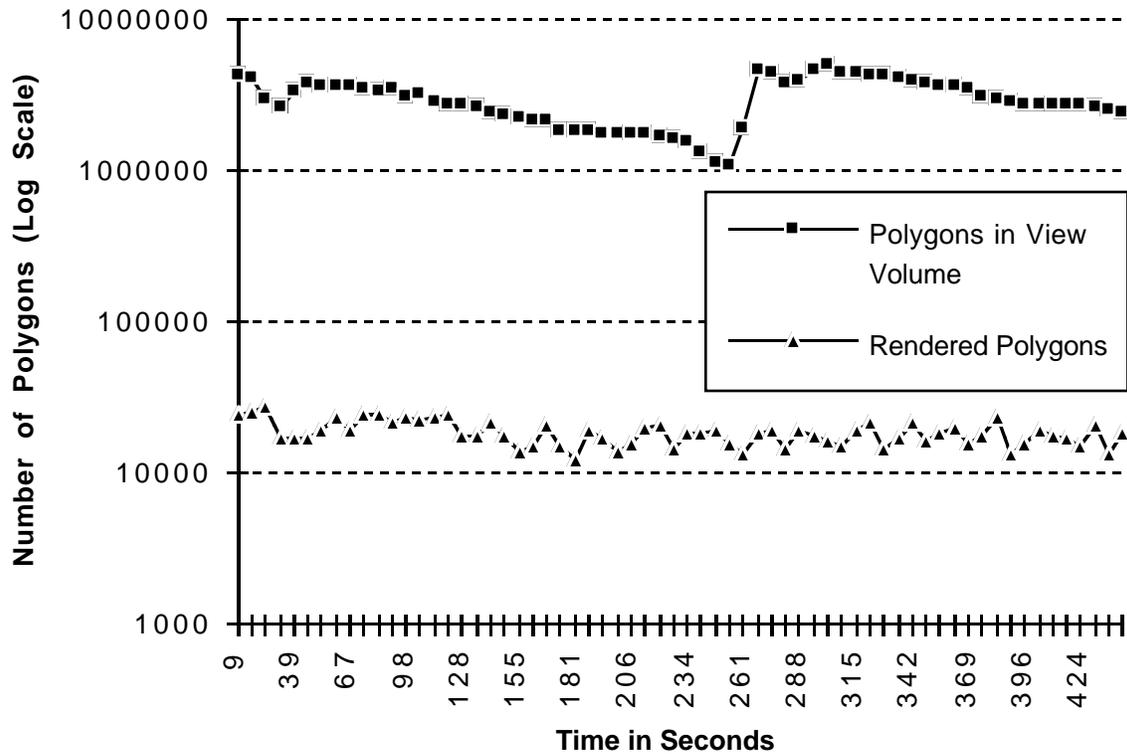
Figure 10 illustrates the difference in polygons with side-by-side wireframe images of the same scene. The image on the right is shown at full resolution. The image on the left is shown after level-of-detail reduction. Most importantly, while the number of bytes of texture and polygons are reduced by one to two orders of magnitude, the perceived image quality is not affected. Figure 11 shows the same scene with texture mapping. On the right is the full resolution polygon mesh and texture. On the left is the same view after level-of-detail reduction in the model. The images are slightly different. But the differences are caused primarily by how the level-of-detail in the detailed model is reduced by the z-buffer and anti-aliasing hardware on the SGI Reality Engine as compared to how the lower level of detail models in our quad tree structure are created.

## 8. SUMMARY

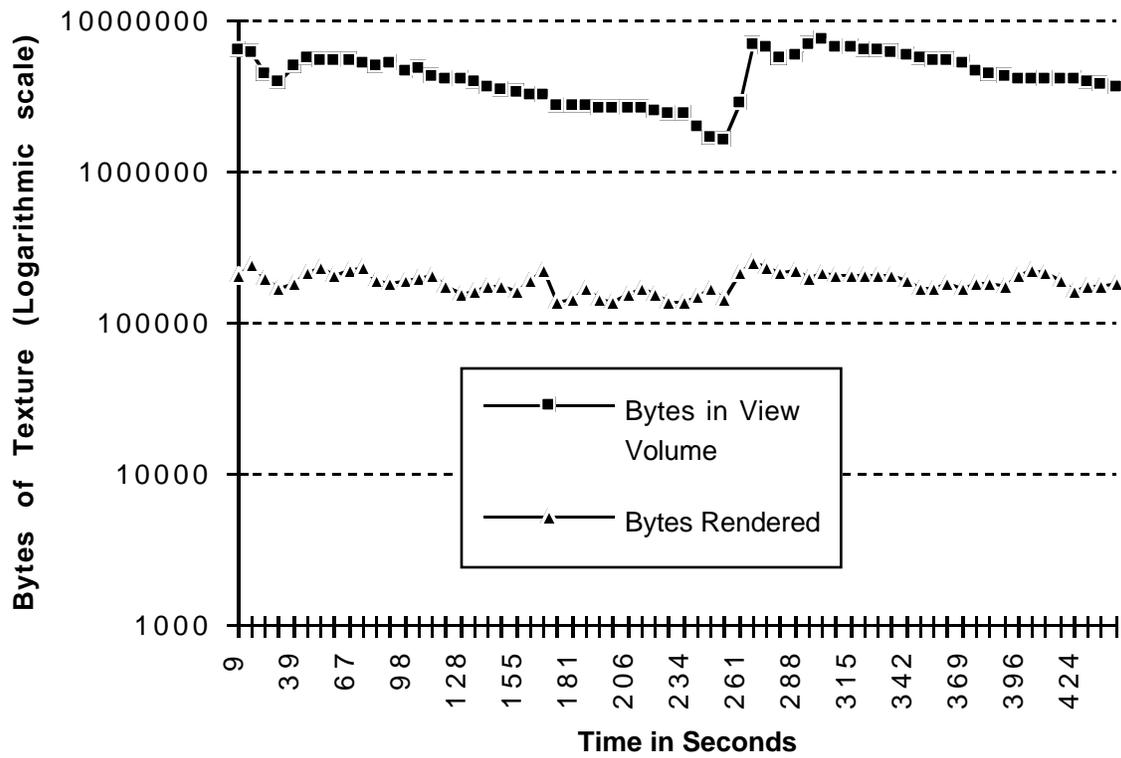
We have described four techniques: distance based polygon reduction, terrain roughness, distance based texture reduction, and texture

viewing angle whose implementation allows real-time level-of-detail management with no loss of detail in the rendered image. Used together, our techniques have reduced the number of bytes of texture used per image by

an order of magnitude, and reduced the number of polygons rendered per frame by two orders of magnitude when compared to full resolution images.



**Figure 8. Comparison of number of polygons rendered to number of polygons in the viewing frustum.** Average frame rate was 17.9 frames/second (s.d.=1.8 frames).



**Figure 9. Comparison of bytes of texture rendered to bytes of texture in the viewing frustum.** Average frame rate was 17.9 frames/second (s.d.=1.8 frames).

## 9 REFERENCES

- Akeley, K. (1993). RealityEngine graphics. *Proceedings of SIGGRAPH 93* (Anaheim, California, August 1-6, 1993), pp.109-116.
- Brown, I., and Pyke, R. (1993). The application of 3-dimensional geographic information systems in geotechnical earthquake engineering. *Proceedings of NSF Workshop on Geographic Information Systems and Their Application in Geotechnical Earthquake Engineering*. Jan 29-30, 1993, Atlanta, Georgia.
- Clark, J.H. (1976). Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19,10, pp. 547-554.
- DeHaemer, M.K., Jr. and Zyda, M.J. (1991). Simplification of objects rendered by polygonal transformations. *Computers & Graphics*, 15,2, pp. 175-184.
- Falby, J.S., Zyda, M.J., Pratt, D.R., and Mackey, R.L. (1993). NPSNET: Hierarchical data structures for real-time three-dimensional visual simulation. *Computers & Graphics* 17,1, pp. 65-69.
- Faust, N. L., Bhaumik, D., Woodward, R., and Vu, D. (1994). Virtual GIS- a new reality. *Proceedings of the International Society for Photogrammetry and Remote Sensing (ISPRS) Commission II Symposium - Systems for Data Processing, Analysis, and Representation*, June 6-10, 1994, Ottawa, Canada.
- Funkhouser, T.A. and Sequin, C.H. (1993). Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Proceedings of SIGGRAPH 93* (Anaheim, CA, August 1-6, 1993), pp. 247-254.
- Heckbert, P.S. and Garland, M. (1994). Multiresolution modeling for fast rendering. *Proceedings of Graphics Interface* (Banff, Alberta 18-20 May, 1994), pp. 43-50.
- Hitchner, L.E and McGreevy, M.W. (1993). Methods for user-based reduction of model complexity for virtual planetary exploration. *Proceedings of SPIE 1913* (San Jose, February, 1993), pp. 1-16.
- Macedonia, M.R. and Zyda, M.J. (1994). NPSNet: a network software architecture for large scale virtual environments. *PRESENCE: Teleoperators and Virtual Environments* 3,4.
- Scarlatos, L. and Pavlidis, T. (1992). Hierarchical triangulation using cartographic coherence. *CVGIP: Graphical Models and Image Processing*, 54,2, pp. 147-161.
- Schroeder, W.J., Zarge, J.A., and Lorensen, W.E. (1992). Decimation of triangle meshes. *Computer Graphics* 26,2, pp. 65-70.
- Williams, L. (1983). Pyramidal parametrics. *Proceedings of SIGGRAPH '83* (*Computer Graphics*, 17,3), pp. 1-11.