Terrain Decimation through Quadtree Morphing

David Cline[1] Parris K. Egbert[2]

Computer Science Department, Brigham Young University

Abstract

We present a new terrain decimation technique called a Quadtree Morph, or Q-morph. The new approach eliminates the usual popping artifacts associated with polygon reduction, replacing them with less objectionable smooth morphing. We show that Q-morphing is fast enough to create a view-dependent terrain model for each frame in an interactive environment. In contrast to most Geomorph algorithms, Q-morphing does not use a time step to interpolate between geometric configurations. Instead, the geometry motion in a Q-morph is based solely on the position of the viewer.

1 Terrain Decimation

A digital elevation model, or DEM, is a rectangular array of height samples taken from terrain data or some other scalar-valued 2D function. One common way to view a DEM is to render a set of triangles that approximates the terrain surface. Unfortunately, the obvious triangulation yields $O(nm)$ triangles, where n and m are the dimensions of the height grid. For large DEMs, this can run into millions of polygons, too many to be rendered in real time even by specialized graphics hardware. *Terrain Decimation* addresses the issue of rendering efficiency by producing a terrain model with significantly fewer triangles that is visually similar to the full resolution model.

While terrain decimation is a widely studied topic, the majority of decimation algorithms in use today suffer from the so called ”popping” artifact. Popping occurs in an animation when successive frames show different approximate models that are visually discontinuous. Decimation algorithms that allow geometric discontinuities between successive approximations are referred to as *discrete* level-of-detail algorithms. Q-morphing avoids popping artifacts by using a *continuous* level-of-detail algorithm instead of a discrete level-of-detail algorithm. Continuous level-of-detail algorithms are able to produce a near infinite number of approximations, such that the visual difference between successive models approaches zero.

2 Previous Work in Terrain Decimation

Several approaches to surface simplification in the setting of terrain models have been proposed. In this section we will discuss four of these, namely, the application of *general polygon decimation* algorithms to terrains, *TIN methods*, *voxelized terrains*, and algorithms that use *semi-regular subdivision*.

2.1 General Polygon Decimation Methods

Although not intended specifically to work on terrain models, methods designed for general polygon decimation can be applied to triangulated DEMs. General methods have the advantage that they can be used for any polygonal objects in the scene, not just terrains. On the other hand, these methods cannot exploit the regular structure of the DEM.

Terrain models have several display characteristics that make some types of decimation algorithms more attractive than others. First, terrain models often consume a large amount of memory. Thus, decimation algorithms that allow progressive transmission of geometry can avert long startup times. Additionally, terrain models are often viewed at close range so that only a small portion of the DEM is visible in any particular view. Thus, algorithms that perform view-dependent simplification are of particular interest in terrain settings.

Luebke and Erikson [15] describe one general method that simplifies complex polygonal environments in a view-dependent fashion. The system works by grouping vertices into hierarchical clusters, and then collapsing clusters into single vertices during animation. Another method that addresses the need for incremental transmission is the progressive mesh [9]. Progressive Meshes, introduced by Hoppe, represent polygonal objects as a small base mesh plus vertex-split transformations. Since the vertex splits can be added incrementally to the model, a progressive representation results. Hoppe later showed how Progressive Meshes can be used for view-dependent simplification, and how they can eliminate popping artifacts by using *Geomorphs* [10].

2.2 TIN Methods

Methods that produce TINs, or Triangulated Irregular Networks, form a large class of terrain decimation algorithms. In general, TIN methods attempt to create a mesh which contains the fewest possible triangles that satisfies some error criteria. Examples of basic TIN methods are found in [5], [21] and [20]. TIN algorithms can produce near optimal results in terms of the number of triangles needed to satisfy a particular error threshold, but most do not operate in real-time.

Since the end goal is often to create a model that will be viewed interactively, TINs can be built off line and stored as triangle meshes. The stored meshes can then be rendered in real-time as long as they contain few enough triangles. One simple approach to detail management in terrains is to use a set of TINs with differing error thresholds as discrete level-of-detail models. Two drawbacks to this approach are that popping can occur when model levels switch, and view-dependent simplification of the terrain is not possible in the individual TINs because the viewpoint is not known at TIN creation time.

Some researchers have suggested extensions to the basic TIN idea that make TINs more suitable for real-time display. Taylor and Barrett [22] address the problem of popping by defining a ”TIN morph” that interpolates between TIN models. Their approach does not address the need for view-dependent model refinement, however. Consequently, the transformations to the terrain geometry must be done on a global scale; local simplification is not possible. Without the ability to simplify local regions in the terrain, the models become too complex for real-time display as the viewer approaches the terrain. DeBerg and Dobrindt [4], and Cohen-Or et al. [3] extend the basic concept of TIN morphing to allow local refinement to the detail level. Both of these algorithms work by

[1]cline@neptune.cs.byu.edu, http://orca.cs.byu.edu/∼cline

[2]egbert@cs.byu.edu

creating hierarchical versions of a Delaunay triangulation. A new tessellation is made for each viewpoint that allows different levels of the hierarchy to coexist on the same model. Another TIN algorithm that uses a hierarchical Delaunay triangulation is discussed by Rabinovich et al. [19]. Besides being able to produce a view-dependent simplification, this algorithm allows piece-wise terrain updates during scene interaction. In more recent work, Hoppe [11] describes a terrain decimation approach that produces terrain models based on a hierarchy of pre-decimated terrain blocks. The system uses run-time geomorphs to eliminate popping artifacts.

2.3 Voxel Methods

Voxel methods do not use polygons to represent the terrain. Instead, the terrain is converted to a grid of discrete voxels, and rendering is done by ray casting. One problem with this approach is that graphics hardware is usually designed to accelerate scan line algorithms–not ray casting. On the other hand, ray casting over terrains lends itself to software optimization based on ray coherence and parallel rendering.

Cohen-Or et al. [2] describe a real-time fly-through engine based on ray casting of a voxelized terrain. Real-time performance is obtained by exploiting ray coherence and rendering in parallel. The system uses a multiresolution version of the terrain similar to a mip-map, and the lower resolution versions of the terrain are substituted for high resolution as the terrain recedes into the distance.

Pagleroni and Petersen [17] use a different approach to accelerate ray casting for voxelized terrains. A virtual cone of empty space is placed on top of each voxel cell, and rays cast into the scene can then skip over the volumes defined by the cones.

2.4 Semi-Regular Subdivision

We classify terrain decimation algorithms that cover the ground plane with 45-45-90 triangles as ”semi-regular subdivision” methods. Like TIN methods, algorithms that use semi-regular subdivision produce triangular models of the terrain. Unlike TIN methods, semi-regular subdivision restricts the placement of vertices in the terrain model to a large degree. Thus semi-regular subdivision cannot expect to achieve the level of optimality obtained using TINs. At the same time, however, the regularity imposed on the models has several advantages. Most importantly, semi-regular subdivision is invariably simpler and runs faster than comparable TIN algorithms. Semi-regular subdivision methods often run fast enough to create a completely new model from the full resolution height grid for each frame of interaction. For this reason view-dependent simplification is the rule rather than the exception for these methods.

For the most part, semi-regular subdivision decimates terrain by subsampling the height grid in a piece-wise fashion. The amount of subsampling for each area of the terrain can be determined by a number of metrics. Falby et al. [7] and Oborn [16] use simple distance cutoffs to determine the level of subdivision. LeClerc and Lau [12] use a metric based on sampling the terrain at a uniform screen frequency. Hitchner and McGreevy [8] use a distance metric augmented by a user-specified ”level of interest” for different parts of the terrain being viewed. Lindstrom et al. [13] base the subdivision level on an estimate of the screen error for small pieces of the terrain.

One approach to semi-regular subdivision that has become widespread in the last few years is the *bintree* or *restricted quadtree triangulation*. First popularized as a terrain decimation algorithm by Lindstrom et al. [14], the bintree approach is based on a particular tessellation of right triangles that allows dense and sparse triangulations to be mixed seamlessly. More recent work has extended the algorithm to allow incremental model updates [6], caching of terrain data and the use of geomorphs to eliminate popping [18].

2.5 The Need for More Research

Despite the large number algorithms extant in the literature to simplify terrains, terrain decimation remains an area of active research. Perhaps the most obvious reason for this continued popularity is the widespread need for fast terrain rendering. Additionally, since the range of datasets and hardware used for terrain rendering varies so widely, implementors faced with a terrain rendering problem often find that previous algorithms lack some essential feature. (In the case of Q-morphing, we needed an algorithm that would mesh well with the texturing framework that we had chosen.) Finally, view-dependent simplification, progressive transmission of geometry, geometry caching and Geomorphs still are not mature disciplines. Thus, more research in these areas is justified.

3 Overview of the Algorithm

A Quadtree Morph builds a view-dependent terrain model by decomposing a DEM into quadtrees. The quadtrees are subdivided so that the final quadtree nodes have a screen size as close as possible to some predetermined value. Figure 1 shows the quadtrees generated for a hypothetical Q-morph. After the quadtree nodes have been chosen, the algorithm subdivides each node into triangles based on an independently-calculated LOD parameter. To prevent cracking, neighboring nodes are forced to have a common boundary. Popping between frames is eliminated using a *position-based* morph.

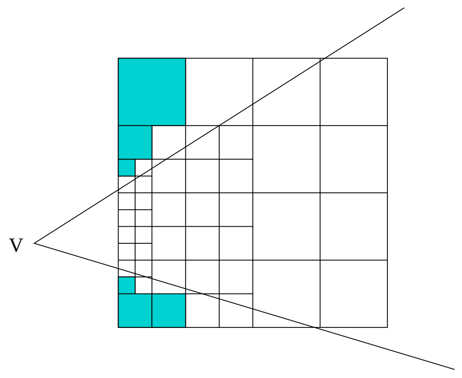

Figure 1: The quadtree nodes of a hypothetical Q-morph. Clipping is done by eliminating quadtree nodes that are outside the view frustum. Subdivision of the quadtrees occurs in a top-down fashion, continuing until the resulting nodes are smaller than a predetermined size in screen space.

3.1 Position-Based Morphing

Most current geomorph methods utilize *time-based* morphing. That is, the rate of geometry motion is controlled by a time step. A *position-based* morph, on the other hand, controls geometry motion by defining a real-valued LOD parameter based on the view position. The integer part of the LOD value determines the base model that will be used, and the mantissa determines the level of morphing. For example, for an LOD parameter of 3.7, the base model would be model 3, and the final geometry would lie seven tenths of the distance between models 3 and 4.

A major advantage of a position-based morph over a time-based morph is that no geometry motion occurs unless the viewer is moving. Additionally, a position-based morph does not have to guess at how much time to allot to a morph. A disadvantage of the position-based morph is that the geometry of the entire object is always in flux, which may slow rendering. Also, it is easier to use a time-based morph to modify individual vertices and edges in a model.

4 Data Structures

In addition to original DEM, the Q-morph algorithm uses three data structures to build a terrain model–the *gridlet quadtrees*, the *temporary DEM* and the *Metagrid*. These are defined in the next sections.

4.1 The Gridlets

The quadtree nodes in a Q-morph, called *gridlets*, correspond to $(2^n + 1) \times (2^n + 1)$ square regions within the DEM that overlap by one sample. We will refer to the smallest gridlets (the leaf nodes of the quadtrees) as *base gridlets*. For the examples presented here, the size of the base gridlets was set to 9×9 samples. We found this to be a good compromise between gridlet memory usage and localization of the screen error.

As mentioned, the gridlets are arranged into a group of quadtrees. The current implementation requires the gridlet quadtrees to be full; however, abutting quadtrees fit together seamlessly as long as the base gridlets have the same dimensions.

When the terrain model is built, a set of gridlets that covers the terrain in the view frustum is chosen based on the viewing parameters. The model is made by subsampling these *final gridlets* independently based on a screen-space error metric. To eliminate popping, a position-based morph is applied to each of the final gridlets.

4.2 The Temporary DEM

The temporary DEM is a "scratch array" of floating point numbers that has the same dimensions as the original height grid. In contrast to the original DEM, which is static, the temporary DEM contains height values that may change at each frame. Because morphing is used to interpolate between approximation levels, the entire terrain model gets rebuilt in the Temporary DEM for each frame.

4.3 The Metagrid

The Metagrid is an array of integers used to coordinate rendering of the gridlet edges. The size of the Metagrid is $A \times B$ where A and B are the dimensions of the DEM in base gridlets.

5 The Algorithm

The process by which Q-morphing builds a view-dependent terrain model can be summarized as follows: 1) determine the list of gridlets that will be drawn, 2) calculate the *step* and *morph* values for the gridlets, and 3) tessellate the gridlets into the temporary DEM. Sections 5.1 to 5.4 discuss these steps in detail. Figure 9 gives pseudo-code for the steps.

5.1 Determining the List of Gridlets to be Drawn

The gridlet screen radius. The gridlet screen radius, r_g, gives an upper bound on the screen size of a gridlet. This value helps determine the list of gridlets that will form the terrain model, and is calculated on a per-gridlet basis by

$$r_g = \frac{h}{\psi} \times \frac{r}{d} \tag{1}$$

where r_g is the gridlet screen radius, h is the height of the viewport in pixels, ψ is the vertical field of view in radians, r the gridlet world radius (1/2 the diagonal of the gridlet bounding box), and d the distance from the viewer to the gridlet center.

The final gridlets. Recursive subdivision is used to determine the final list of gridlets that will be drawn. Starting with the root nodes of the gridlet quadtrees, the algorithm descends until either the gridlet is outside the view frustum and can be culled, or the screen radius falls below some threshold, r_{max}. The choice of r_{max} is a vital part of the algorithm. If it is too small, an abundance of final gridlets will result, slowing the rendering process because of gridlet overhead. On the other hand, if r_{max} is too large, the screen error will not be localized well, and poor quality or a high polygon count will result. Our studies indicate that values of r_{max} between about 60 and 120 pixels work well for standard sized graphics screens.

5.2 The Step and Morph Values

Each of the final gridlets is constructed based on two numbers called the *step value*, s_f, and the *morph value*, m_f. The step value determines the amount of subsampling used to reconstruct the gridlet, and the morph value determines the extent of interpolation between subsampling levels. Section 5.4 explains in detail how the step and morph values are used to make the terrain model.

s_f and m_f are defined in terms of a *final LOD value*, t_f. Given t_f, the step and morph values are calculated as follows:

$$s_f = 2^{\lfloor t_f \rfloor}$$

$$m_f = t_f - \lfloor t_f \rfloor \tag{2}$$

5.2.1 The Linear LOD Value

Each gridlet traversed during the Q-morph calculates a *linear LOD value*, t_l, that encodes the reconstruction quality needed to satisfy a particular error threshold. t_l is called *linear* because it maps the reconstruction quality onto a linear scale. A value of zero corresponds to a perfect reconstruction, and higher values successively decrease in fidelity. Several variables are used in the computation of t_l, namely the *height disparity list*, the *screen disparity multiplier*, and the *screen error tolerance*.

The height disparity list. Each gridlet contains a height disparity list, H, for its region of the DEM. H is used to estimate the terrain roughness at differing resolutions. H_i is defined as the maximum, or mean, vertical disparity between tessellating the gridlet at full resolution and subsampling at resolution 2^i. In practice, the definition is modified as follows to prevent popping and rapid morphing artifacts:

$H_0 = k_{h0} H_1$.[3]
For $i > 1$, $(H_i - H_{i-1}) \geq k_r(H_{i-1} - H_{i-2}) + k_a$.[4]

[3]We used a value of $1/3$ for k_{h0}.

[4]k_a assures that H is strictly increasing. k_r makes sure that morphing does not take place too quickly.

The screen disparity multiplier. Before the LOD value for a gridlet can be determined, the vertical disparities in H must be converted to screen space. Q-morphing approximates this operation by calculating a screen disparity multiplier, D_s, for each gridlet. D_s converts vertical disparity in height units to screen error in pixels, and is calculated by

$$D_s = \frac{h}{\psi} \times \frac{\sin\alpha}{d} \tag{3}$$

where h is the viewport height in pixels, ψ is the vertical field of view angle in radians, α is the angle between the terrain "Up" vector and the vector from the gridlet center to the viewpoint, and d is the distance from the viewpoint to the gridlet center.

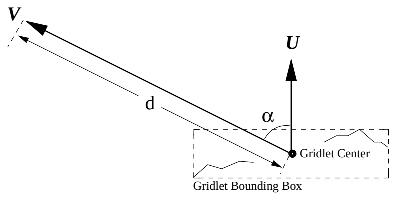

Figure 2: The screen disparity multiplier estimates the screen space error of one unit of vertical disparity on a per-gridlet basis.

The screen error tolerance. Q-morphing relies on a user-specified screen error tolerance to determine the amount of error allowed in the finished model. Depending on the definition of the height disparity list, the error tolerance is interpreted as either the approximate average or maximum deviation in screen space between the desired model and the Q-morph model.

Calculation of the linear LOD value. Let e be the screen error tolerance in pixels and let $i = \log_2(GridletSide - 1)$. T_l is calculated by mapping e onto the height disparity list scaled by D_s. Figure 3 shows the mapping. Equation 4 defines t_l formally.

$$t_l = \begin{cases} 0 & \text{if } e < D_S H_0 \\ i & \text{if } e > D_S H_i \\ otherwise & \\ (j-1) + \frac{e - H_{j-1} D_s}{(H_j - H_{j-1}) D_s} & D_s H_{j-1} < e \leq D_s H_j \end{cases} \tag{4}$$

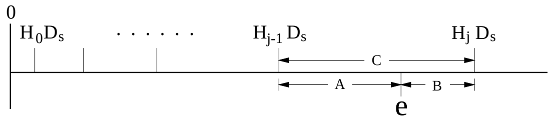

Figure 3: The linear LOD value is defined by mapping e onto the height disparity list, H, scaled by D_s. Applying equation 4, t_l for the example shown in the figure is $((j-1) + A/C)$.

5.2.2 The Final LOD Value

As explained in section 5.1, when the screen radius of a gridlet exceeds the value r_{max}, the children of the gridlet are used for rendering. Since LOD values calculated for the child and parent gridlets are in general different at the transition between levels, popping could result. To eliminate this popping, the *final LOD value*, t_f, is interpolated between *linear LOD values* that are calculated for the parent and child gridlets. Conceptually, t_f is a weighted sum of the linear LOD values of the gridlet to be drawn and its parent. t_f is computed as:

$$t_f = m t_g + (1-m) t_p \tag{5}$$

where

$$m = \frac{\frac{1}{1-B}}{\frac{1}{1-B} + \frac{1}{1-A}}$$

$$A = r_{max}/r_p$$
$$B = r_g/r_{max}$$

$t_f =$	The final LOD value.
$t_p =$	Linear LOD value of parent gridlet.
$t_g =$	Linear LOD value of gridlet to be drawn.
$r_p =$	Screen radius of parent gridlet.
$r_g =$	Screen radius of gridlet to be drawn.

Of course, t_g is used as the final LOD value if the gridlet to be drawn has no parent, or if $r_g > r_{max}$.

5.3 Setup Procedures

Before the height values in the temporary DEM can be set, several setup steps must be done to coordinate gridlet rendering. These steps are sorting the gridlets, clearing the gridlet borders and setting values in the Metagrid.

Sorting the gridlets. The gridlets are sorted in descending order of final LOD value to coordinate edge drawing. Without the sorting, gaps would be introduced on gridlet edges.

Clearing the Gridlet Borders Prior to building the terrain model, the final gridlet borders are cleared in the temporary DEM using s_f, defined in section 5.2, as a step value. When the terrain model gets built, the border values will be set only once, and this will eliminate tears between gridlets. However, it is still possible for T-junctions to occur. Although we have not taken steps to eliminate the minute cracks caused by the T-junctions, they are usually not noticeable on textured terrain. Moreover, the cracks could be eliminated fully by adding a few triangles to weld the gridlet edges together.

Setting the values in the Metagrid. To coordinate rendering, the final gridlet step values (s_f) are copied to the Metagrid. When a gridlet is processed, the smallest neighboring step value on each side is used to determine if interpolation is necessary. Interpolation forces neighboring girdlets to a common edge, eliminating cracking.

5.4 Making the terrain model

The terrain model is built by tessellating the final gridlets into the temporary DEM. To process a gridlet the algorithm needs the step value for the gridlet, s_f, the morph value, m_f, and the minimum step values of the neighboring gridlets, s_l, s_r, s_t and s_b.

Setting the edges. As mentioned, the gridlet edges are processed in decreasing order of final LOD value to avoid introducing gaps into the model. Since a vertex in the temporary DEM is set only once, the height calculation of an edge vertex is bypassed if a value for that location has already been set in the temporary DEM.

The edges of the final gridlets are processed as follows: First, edge vertices with coordinates that are even multiples of s_f are copied directly from the original DEM. Next, edge vertices lying on odd multiples of s_f are morphed using the equation

$$t_o = m_f \frac{t_{-sf} + t_{sf}}{2} + (1 - m_f)h_o \tag{6}$$

where t_o is the computed height of the vertex at an odd position, m_f is the morph value for the gridlet, t_{-sf} and t_{sf} are the heights of the neighboring vertices in the temporary DEM, and h_o is the height of the vertex as specified in the original DEM. Note that while h_o comes from the original DEM, t_{-sf} and t_{sf} are taken from the temporary DEM. A special case occurs when s_f is the same as the gridlet width. In this case the gridlet corners must be morphed using values outside the gridlet boundaries. In all other respects, however, the calculation proceeds normally. After the morphed vertices are set, points that lie on step values of neighboring gridlets are set by linear interpolation. Figure 4 shows the calculation of the gridlet edges in the terrain model.

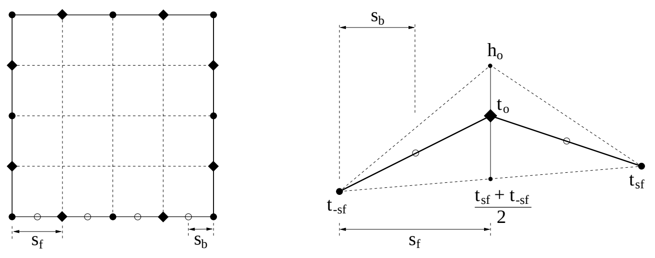

Figure 4: Computation of the gridlet edges. Height values for vertices on even multiples of s_f are copied directly from the original DEM (filled circles). Vertices on odd multiples of s_f (diamonds) are computed using equation 6. Vertices lying on neighboring step values are set by linear interpolation (hollow circles).

The gridlet interiors. Vertices interior to the gridlets are set similar to the edges. Points having x and y coordinates that are both even multiples of s_f are copied from the original DEM. Other points are calculated using equation 6. Figure 5 shows the morphing directions used in the gridlet interior, that is, the directions to the vertices used as t_{-sf} and t_{sf}.

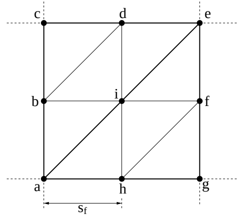

Figure 5: Morphing directions for a 3 × 3 section in the interior of a gridlet. Vertices a, c, e, and g are copied from the original DEM. Other vertices are calculated using equation 6 with b and f being morphed vertically, d and h morphed horizontally, and i morphed diagonally between a and e.

6 Drawing the Terrain Model

Triangle strips. Our current implementation renders the terrain model as a set of triangle strips. Each horizontal strip of polygons in a gridlet is turned into one triangle strip.

Large textures To create a convincing terrain simulation, detailed texture must be added to the polygonal model created in a Q-morph. We use the caching algorithm described in [1] to manage this detail. Since the texture caching scheme and Q-morphing both use a quadtree as the underlying structure, the geometry and texture can be aligned so that each gridlet covers exactly one texture tile. Furthermore, since Q-morphing bounds the screen size of the final gridlets, texture LOD decisions can be made on a per-gridlet basis.

7 Results

To test the Q-morphing algorithm, we used a dataset of the Wasatch Front in northern Utah containing several sites that will be used in the 2002 Winter Olympics. The original DEM for the Wasatch Front dataset is 1601 × 3073 samples in size with 31.875 meters between samples. The DEM for the scene is divided into 76,800 9 × 9 base gridlets, and contains 102,357 gridlets in total.

7.1 Number of Triangles

We performed several flybys of the Wasatch Front dataset using different error thresholds to determine the correlation between the error threshold and the number of triangles in the final model. Figure 6 shows the number of triangles generated by three flybys using the same camera path but different error thresholds. Note the strong similarity that exists between the shapes of the curves in the graph. This indicates a multiplicative relationship between the error threshold and the resulting number of triangles.

Figure 7 gives the framerate sustained for a flyby of the South Wasatch dataset by an SGI Reality Station with a single R10000 processor.

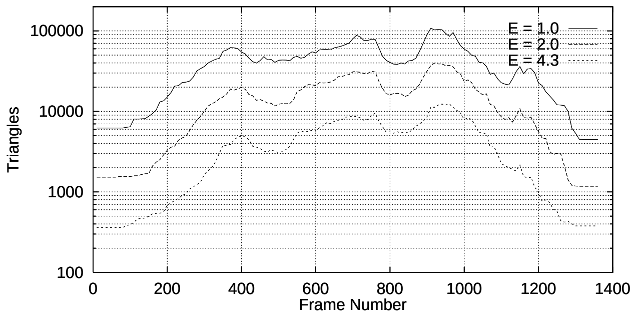

Figure 6: The number of triangles generated by a Q-morph under different error thresholds. Results are calculated for a flyby of the Wasatch Front dataset with error thresholds of 1.0, 2.0 and 4.3 pixels on a 640 × 480 viewport.

7.2 Framerate

As can be seen in figures 6 and 7, the number of polygons produced by a Q-morph, and the framerate in the resulting animation, is not at all constant. Real-time applications require algorithms that complete within a specified amount of time, however. For modern graphics hardware, constant time often equates to constant polygon count. Thus, a useful extension to Q-morphing would be to allow

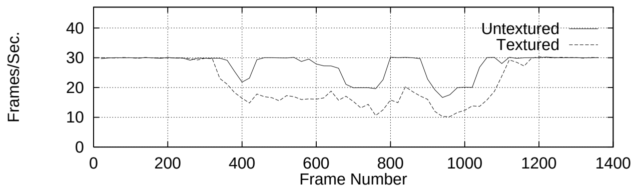

Figure 7: Framerates achieved by the Q-morphing algorithm. The screen size for this trial was 640×480 and the error threshold was 2.0 pixels.

the error threshold to float to produce models having a near constant number of polygons.

The current implementation achieves a near constant polygon count by using the polygon count in an animation frame to steer the error threshold used for the next. For example, if the number of polygons rendered in a given frame is greater than the target number, the error threshold for the next frame is increased by a small epsilon, say 1/10 of a pixel. Conversely, if the number of polygons is less than the target, the error theshold is decreased by the same epsilon. This simple solution works well in most situations, but can be prone to slope overloading during fast animation sequences.

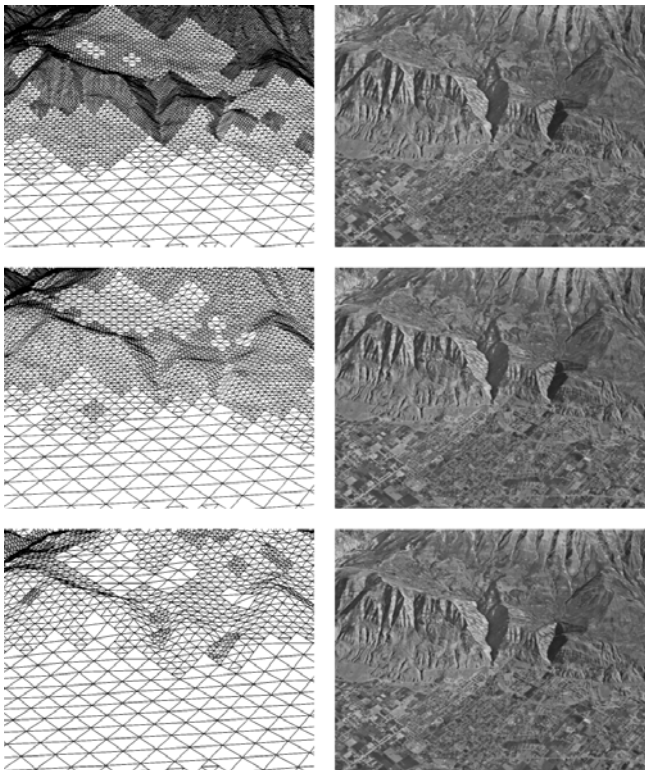

Figure 8: Results of the Q-morphing algorithm. The images shown were captured from a 900×700 screen. Left images show the models produced using different error thresholds. The images on the right are textured versions of the same models. From top to bottom, the error threshold is 1.0, 2.0 and 4.3 pixels. The models have 28460, 10520 and 3536 triangles respectively.

7.3 Visual Results

Figure 8 shows Q-morphs of the South Wasatch dataset made with different error thresholds. Although the number of triangles in the different models varies by a factor of eight, the textured versions of the scene are almost identical one-to-another.

8 Conclusion

In this paper we have presented a new terrain decimation algorithm called Q-morphing. We have shown that the Q-morph algorithm can produce high quality, view-dependent terrain models based on original DEM data, and that these models can be rendered at interactive frame rates. Additionally, we have shown that by using a geomorph technique based on the view position, a Q-morph completely eliminates popping artifacts, producing a more believable simulation.

9 Future Work

Incremental model updates. Q-morphing makes a completely new terrain model for each frame of interaction. If terrain model creation time is is an issue, the algorithm could take advantage of frame coherence to incrementally update the model instead of starting from scratch each frame.

Terrain caching. Currently, Q-morphing requires that all of the original DEM be in main memory. If Q-morphing is to be applied to very large terrains, it must be modified to allow caching of terrain data and the gridlet support structures.

References

[1] David Cline and Parris K. Egbert. Interactive Display of Very Large Textures. In *IEEE Visualization 1998*, pages 343–350, 1998.

[2] Daniel Cohen-Or et al. A Real-Time Photo-Realistic Visual Flythrough. *IEEE Transactions on Visualization and Computer Graphics*, pages 255–265, 1996.

[3] Daniel Cohen-Or and Yishay Levanoni. Temporal Continuity of Levels of Detail in Delaunay Triangulation. In *IEEE Visualization*, pages 37–42, 1996.

[4] M. DeBerg and K. T. G. Dobrindt. On Levels of Detail in Terrains. In *11th ACM Symposium on Computational Geometry*.

[5] L. DeFloriani and E. Puppo. Hierarchical Triangulation for Multiresolution Surface Description. *ACM Transactions on Graphics*, 14(4), October, 1995.

[6] Mark Duchaineau et al. ROAMing Terrain: Real-time Optimally Adapting Meshes. *IEEE Transactions on Visualization and Computer Graphics*, pages 81–88, 1997.

[7] John S. Falby et al. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers and Graphics*, 17(1):65–69, 1993.

[8] L. E. Hitchner and M. W. McGreevy. Methods for User-Based Reduction of Model Complexity for Virtual Planetary Exploration. In *Proceedings of the SPIE*, volume 1913, pages 1–16, 1993.

[9] Hughes Hoppe. Progressive Meshes. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 99–108. ACM SIGGRAPH, 1996.

[10] Hughes Hoppe. View-Dependent Refinement of Progressive Meshes. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 189–198, 1997.

[11] Hughes Hoppe. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. In *Proceedings IEEE Visualization 1998*, pages 35–42, 1998.

[12] Y. G. Leclerc and S. Q. Lau. TerraVision: A Terrain Visualization system. *Technical Note 540, SRI International, www.sri.com*, pages 1–20, 1994.

[13] Peter Lindstrom et al. Level-of-Detail Management for Real-time Rendering of Phototextured Terrain. *GIT-GVU Technical Report*, 95-06.

[14] Peter Lindstrom et al. Real-Time Continuous Level of Detail Rendering of Height Fields. In *Computer Graphics*, pages 109–118. ACM SIGGRAPH, 1996.

[15] David P. Luebke and Carl Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. In *Computer Graphics (SIGGRAPH '97 Proceedings)*.

[16] Shaun M. Oborn. UTAH: The Movie. *Master's Thesis, Utah State University, Logan UT*, 1994.

[17] D. S. Pagleroni and S. M. Petersen. Height Distributional Distance Transform Methods for Height Field Ray Tracing. *ACM Transactions on Graphics*, 13(4):376–399, October 1994.

[18] Renato Pajarola. Large Scale Terrain Visualization Using The Restricted Quadtree. In *Proceedings IEEE Visualization 1998*, pages 19–26, 1998.

[19] Boris Rabinovich and Craig Gotsman. Visualization of Large Terrains in Resource-Limited Computing Environments. *IEEE Transactions on Visualization and Computer Graphics*, pages 95–102, 1997.

[20] L. Scarlatos and T. Pavlidis. Hierarchical triangulation using cartographic coherence. *CVGIP: Graphical Models and Image Processing*, 54(2):147–161, 1992.

[21] F Schroder and P. RossBach. Managing the Complexity of Digital Terrain Models. *Computers and Graphics*, 18(6):65–70, 1994.

[22] David C. Taylor and William Barret. An Algorithm for Continuous Resolution Polygonalizations of a Discrete Surface. In *Proceedings of Graphics Interface '94*, pages 33–42, 1994.

```
GridletList S;    // contains the root nodes of the gridlet quadtrees.
GridletList F;    // starts empty.


BuildTerrainModel( )
{
    for ( i=0; i < S.size; i++ )
        Process( S[i], NO_PARENT_FLAG );

    Sort F in descending order by final LOD value.

    for ( i=0; i < F.size; i++ )
        Calculate border height values for F[i].    // fig. 5 and eq. 6
        Calculate interior height values for F[i].    // fig. 5
        Draw F[i].
}


Process( Gridlet G, float parentLOD )
{
    float gridletScreenRadius;
    float linearLOD;

    if ( G is outside the view frustum )
        Write GRIDLET_CULLED flag to Metagrid.
        return;

    Calculate gridletScreenRadius for G.    // eq. 1
    Calculate linearLOD for G.    // fig. 3 and eq. 4

    if ( gridletScreenRadius > $r_{max}$   AND   G has children )
        for ( i=0; i<4; i++ )
            Process( G.child[i], linearLOD );
        return;

    Add G to F.
    Calculate finalLOD for G using parentLOD and linearLOD.    // eq. 5
    Calculate morphValue and stepValue for G.    // eq. 2
    Write stepValue of G to Metagrid.
    Clear borders of G in Temporary DEM subsampling by stepValue.
    return;
}
```

Figure 9: Pseudo-code of a Q-morph. Before calling BuildTerrainModel, **S** must contain the root nodes of the gridlet quadtrees, and **F** must be empty. After BuildTerrainModel returns, **F** will contain the final gridlets that make up a view dependent terrain model. The height values in the temporary DEM will also have been updated.