Real-Time Generation of Continuous Levels of Detail for Height Fields

Stefan Röttger (snroettg@immd9.uni-erlangen.de)
Wolfgang Heidrich, Philipp Slusallek, Hans-Peter Seidel

Graphische Datenverarbeitung (IMMD9)
Universität Erlangen-Nürnberg

ABSTRACT

Height fields play an important role in the fast growing domain of Geographic Information Systems (GIS). For exploring different kinds of geographic-based data sets on screen it is necessary to display height fields at interactive frame rates. Because of the inherent geometric complexity, this goal is often unachievable even with new generations of powerful graphics computers, unless the original height field data is approximated in order to reduce the number of geometric primitives that need to be rendered without compromising visual quality.

So far most algorithms have focused on global reduction or multi-resolution techniques, which reduce resolution on the basis of surface roughness. A recent new approach called *Continuous Levels of Detail* [LKR+96] introduced a hierarchical quadtree technique. In order to reduce the projected pixel error, the height field is dynamically triangulated in a bottom up fashion according to the distance to the point of view. Since resolution is allowed to change smoothly, the result is a much better image quality.

However, this algorithm still has a major disadvantage. With the viewpoint moving, the triangulation is continuously changing, resulting in a phenomenon called *vertex popping*. As the observer approaches an area with detail information, this detail will suddenly appear at a certain distance. To eliminate these artifacts we introduce a new, rapid geomorphing algorithm, which operates top down on a quadtree data structure.

1 Introduction

A fast growing domain in computer graphics are the so called Geographic Information Systems (GIS). They allow to explore large geographic data sets interactively on screen, which involves displaying height fields in real-time. Typical height fields consist of a large number of polygons, so that even most high performance graphics computers have great difficulties to display even moderately sized height fields at interactive frame rates. The common solution is to reduce the complexity of the scene while maintaining a high image quality.

Most existing algorithms work on general surfaces by building a lower resolution mesh [GGS95] or a Triangulated Irregular Network (TIN) [TB94]. A height field is triangulated by taking its roughness into account, building on the fact that flat areas and smooth regions can be approximated by fewer triangles than rough regions. On the other hand, height fields have special properties, that one should take advantage of for reducing the geometric complexity even further. When displaying a height field, there will almost always be both regions that are quite close to the point of view and those that are far away. As with surface roughness, close regions must be approximated more accurately than regions that are far away. Furthermore, as the viewpoint is moving, the triangulation no longer remains static. The term *Level of Detail* stands for all algorithms that exploit this property.

A well known technique in that domain are the so called Progressive Meshes [Hop96]. With recent additions [Hop97], this techniques can also be applied to view dependent triangulations, but requires large data structures.

There are also various real-time simulation systems, which divide height fields into smaller blocks, and generate several multi-resolution triangulations for each of those [SN95, KLR+96]. Switching between these different levels of detail is done depending on system stress and distance to the point of view.

Several problems need to be solved with this technique: First of all, cracks must be avoided between adjacent edges of blocks at differing resolution. In addition, popping must be handled when replacing a block with one from another level of detail. The visual appearance of the popping effect can be eliminated by geomorphing between both levels of detail. However, this multi-resolution strategy is not optimal, since it is assumed that the distance to the point of view is constant throughout each block.

An algorithm specifically designed for height fields was presented at Siggraph '96 [LKR+96]. It uses a dynamically changing quadtree and a bottom up strategy to determine whether a node has to be subdivided or should be merged with adjacent nodes. For that purpose it calculates an upper bound on the projected pixel error, which is taken to be an on screen error measure for image quality. The main disadvantage of this bottom up strategy is that the pixel error function has to be evaluated for all points of the height field. That would be very costly, unless an error interval is computed, which avoids subdivision and merging for a large number of vertices. If modifications of the triangulation are necessary, all affected nodes are visited. In that case all adjacent nodes also have to be updated in a bottom up fashion. This results in a view-dependent triangulation that allows for smooth transitions between different points of view. Although it would theoretically be possible to include geomorphing in this algorithm, this is not implemented in the current version, so that popping still occurs.

We now present an algorithm that uses a top-down strategy to create a triangulation and exploits geomorphing at virtually no additional cost. Vertex removal is performed depending on its distance to the point of view as well as local surface roughness, which is pre-calculated. Using a top-down approach we only need to visit a fraction of the whole data set in each frame, which allows for high frame rates even with large height fields. On the down side, more involved criteria such as silhouette tests cannot be included into this method, since they require the analysis of the whole data set for each view point. In practice, however, this is not too restrictive, because over-emphasizing silhouettes causes lighting artifacts for non-silhouette polygons.

The quadtree structure of our method allows for very efficient clipping. Furthermore, memory usage is limited to the space required for the height field data, the texture map plus one additional byte per data point. In the following we incrementally develop our top-down method, which includes geomorphing in a natural fashion.

2 The Algorithm

The underlying data structure of the algorithm is basically a quadtree. For the discussion in this paper, we assume that the height field are of size $2^n + 1 \times 2^n + 1$. A sample triangulation generated by our algorithm is shown in Figure 1.

The quadtree is represented by a boolean matrix with each block's center entry set, if the corresponding node is further refined. The quadtree matrix of the example mesh in Figure 1 is shown below:

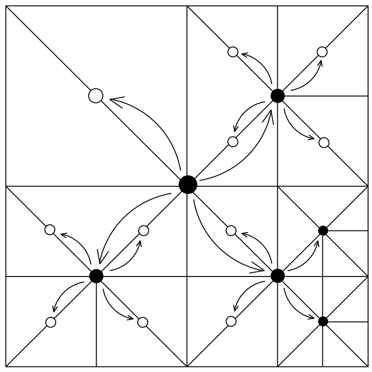

Figure 1: A sample triangulation of a 9×9 height field. The arrows indicate parent-child relations in the quadtree.

$$\begin{pmatrix}
? & ? & ? & ? & ? & ? & ? & ? & ? \\
? & ? & ? & ? & ? & 0 & ? & 0 & ? \\
? & ? & 0 & ? & ? & ? & 1 & ? & ? \\
? & ? & ? & ? & ? & 0 & ? & 0 & ? \\
? & ? & ? & ? & 1 & ? & ? & ? & ? \\
? & 0 & ? & 0 & ? & 0 & ? & 1 & ? \\
? & ? & 1 & ? & ? & ? & 1 & ? & ? \\
? & 0 & ? & 0 & ? & 0 & ? & 1 & ? \\
? & ? & ? & ? & ? & ? & ? & ? & ?
\end{pmatrix}$$

Matrix entries labeled with a question mark do not have to be set during the calculation of the triangulation, since these values are not accessed by the top-down algorithm for the given triangulation. Because the number of nodes that have to be visited for each frame only depends on the rendering quality, but not on the height field size, the required memory bandwidth is limited by the desired image quality.

2.1 Rendering the Height Field

The triangulated height field is drawn by recursively traversing the quadtree where the corresponding matrix entries are set. Whenever a quadtree leaf is reached, a full or partial triangle fan [Boa92, NDW93] is drawn. Triangle fans are well suited for drawing triangulations with varying resolutions: In order to avoid gaps at places where adjacent blocks have different resolution, a conforming mesh is generated simply by skipping the center vertex at these edges (see Figure 2). This method works as long as the levels of adjacent sub-nodes differ by no more than 1. At the end of the next section, we will see how this requirement can be maintained during rendering by preprocessing the height field and storing surface roughness information.

During the generation of the triangle fans we need to determine whether adjacent nodes are subdivided to the same level, or not. If the neighboring node is not subdivided to the same level, we can skip the center vertex on the shared edge. This case can be detected by checking the matrix entry corresponding to the neighboring node, which then has to be zero (Note, that accessing matrix entries, that have not been set, is excluded, since level differences are supposed to be less than or equal to one).

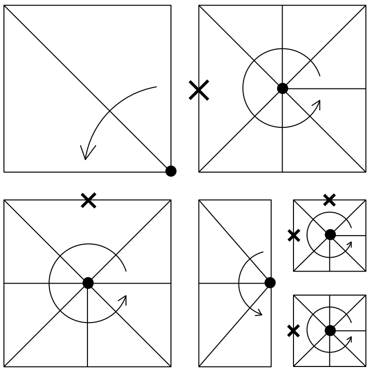

Figure 2: Recursively generated triangle fans for the triangulation shown in Figure 1. The crosses indicate skipped vertices.

2.2 Generating the Triangulation

Before a scene can be rendered as described in the last section, the triangulation has to be built by recursively descending the quadtree. At each subnode a boolean subdivision criterion is evaluated and its result is stored in the quadtree matrix. If the condition is true and the finest level of detail has not yet been reached, we descend further down the tree by visiting all four sub-nodes.

Several aspects need to be taken into account for the criterion: First of all, the resolution should decrease as the distance from the viewer increases. This condition can be guaranteed by ensuring that

$$\frac{l}{d} < C \tag{1}$$

for some constant C, where l is the distance to the eye point, and d is the edge length of the block (see Figures 3 and 4). C is a configurable quality parameter.

The constant C controls the minimum global resolution. As C increases, the total number of vertices per frame grows quadratically. Note that the condition is evaluated only once for a complete triangle fan, which consists of up to 10 vertices. In order to allow for efficient computations, distance measurement is performed using the L^1-norm.

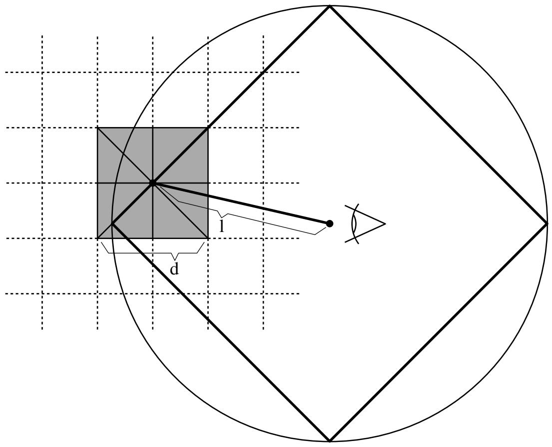

Figure 3: Global resolution criterion: distance versus size of quadtree cells.

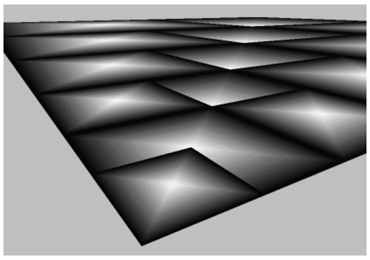

Figure 4: Triangulation of flat geometry based on the global resolution criterion. Centers of triangle fans have been colored white and edges black.

With the second criterion we want to increase the resolution for regions of high surface roughness. In fact, we want to minimize the projected pixel error, which is a good measure for image quality. When dropping one level of the hierarchy, new error is introduced at exactly five points: at the center of the quadtree node and the four midpoints of its edges. An upper bound to the approximation error in 3-space can be given by taking the maximum of the absolute values of the elevation differences dh_i (see also Figure 5). The elevation differences are computed along the edges of the node, as well as along its diagonals, which makes a total of six values per node. The error in 3-space introduced by dropping one level in the quadtree can now be computed by pre-calculating the maximum of the absolute values of these elevation differences, or alternatively by pre-calculating surface roughness values, which we call $d2$:

$$d2 = \frac{1}{d} \max_{i=1..6} |dh_i| \tag{2}$$

The $d2$-values of a node times the edge length d of the node correspond to the approximation error in 3-space. Thus, the $d2$-value times d is an upper bound for the error introduced by dropping one level of detail.

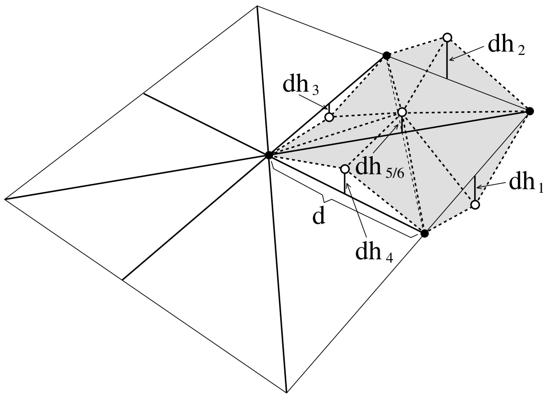

Figure 5: Measuring surface roughness

A revised version of the subdivision criterion (1) which includes the $d2$-values for handling surface roughness can now be given in terms of a decision variable f:

$$f = \frac{l}{d \cdot C \cdot \max\left(c \cdot d2, 1\right)} \qquad (3)$$

$$\texttt{subdivide if } f < 1$$

The constant C again determines the *minimum* global resolution, whereas the newly introduced constant c specifies the *desired* global resolution. The latter constant directly influences the number of polygons to be rendered per frame. Thus, by adjusting c to the current system load, a constant frame rate can be maintained.

The major issue that remains open is how to guarantee that the level difference of adjacent blocks is less than or equal to one. Since the surface roughness of adjacent blocks may differ significantly, this is necessary to build a conforming mesh without holes. In the following we describe how this can be achieved.

First suppose that Condition (3) is true for a given block ($f_2 < 1$), that is, the block has to be subdivided. In this case, all adjacent blocks of twice the edge length have to be subdivided, too. Thus, the following condition must hold for the decision variable f_1 of an adjacent block in order to limit the level differences:

$$f_1 < f_2 \Leftrightarrow \frac{l_1}{d \cdot d2_1} < \frac{l_2}{\frac{d}{2} \cdot d2_2} \qquad (4)$$

For a point of view falling inside the rectangular region (indicated in Figure 6) Equation (3) is always satisfied, since $\frac{l_1}{d}$ is always less than the minimum resolution C. Outside this region the value of the fraction $\frac{l_1}{2l_2}$ is bounded by $\frac{1}{2}$ (for an infinitely distant point of view) and the constant K with:

$$\frac{1}{2} < \frac{l_1}{2l_2} < K \quad (C > 2) \qquad (5)$$

$$K = \frac{L_1}{2L_2} = \frac{C}{2\left(C-1\right)}$$

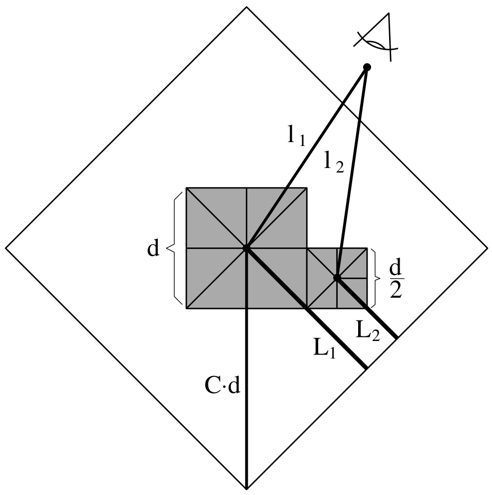

Figure 6: Constraints on $d2$-values of adjacent blocks in order to satisfy Condition (4).

In other words, if $\frac{d2_1}{d2_2}$ is greater than K, then Condition (4) is true, since $\frac{l_1}{2l_2}$ satisfies Condition (5). However, since the $d2$-values, which correspond to surface roughness, can grow arbitrarily large, Condition (4) is not automatically fulfilled. Thus, if $\frac{d2_1}{d2_2} \leq K$, then we have to modify the $d2$-values in the following fashion: Starting with the smallest existing block, we calculate the local $d2$-values of all blocks and propagate them up the tree. The $d2$-value of each block is the maximum of the local value and K times the previously calculated values of adjacent blocks at the next lower level. For our example, the $d2$-values propagated from the bottom to the top of the quadtree are shown in Figure 7.

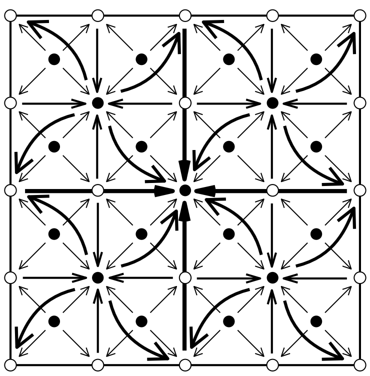

Figure 7: $d2$-values are propagated from bottom up (indicated by arrows).

So far, we have only considered the 2D case, but with some care we can adopt it for 3D. In this case, the elevation of the view point needs to be taken into account relative to the center of quadtree cells. However, since height fields usually have small elevation compared to their size, this distance can be approximated by the difference between the elevation of the view point and the average elevation of the quadtree nodes.

An example of the influence of the propagation of $d2$-values throughout the height field, and its impact on the triangulation is given in Figure 8. Here a few small peaks are placed on an otherwise flat surface.

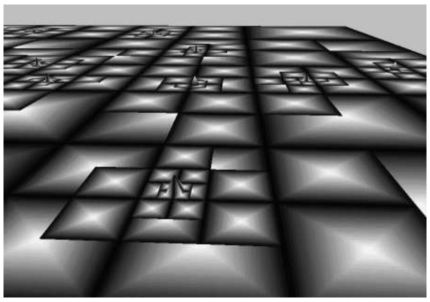

Figure 8: Propagating $d2$-values causes finer triangulation near local peaks in a flat surface.

2.3 Geomorphing

So far, we have shown how to triangulate and render a height field, but with a changing point of view popping still occurs. Remember the decision variable f from Equation (3): A close examination reveals that, if f falls into the range $[\frac{1}{2}, 1)$, the quadtree is not further refined, and a single triangle fan is generated for the complete node. Values of less than $\frac{1}{2}$ indicate that the node has at least one child, while for values larger than 1, the node has no child at all.

Since morphing only happens in the leaf nodes of the current triangulation, we can use $b = 2\,(1 - f)$ clamped to the range $[0, 1]$ as a blending factor to morph between two levels of detail (see Figure 5). Depending on how deep adjacent quadtree nodes are subdivided, there are up to five vertices where morphing might have to be performed for each quadtree node (see Figures 2 and 6). The elevation at these points is interpolated linearly with factor b between the elevation of the lower level (which is the average of the two corresponding corner points) and the elevation of the higher level. The latter is taken directly from height field data.

Some caution is required for avoiding cracks and generating a conforming mesh. Blending factors of adjacent blocks differ slightly due to a variation of the distances to the point of view. Thus, the interpolated elevation at the midpoint of a shared edge is different for adjacent blocks, causing cracks to appear.

In order to avoid this, we store blending factors in the matrix, rather than boolean values. The blending value for a shared edge is obtained by taking the minimum of the blending values of the two involved blocks. A value of zero indicates no subdivision, while other values directly represent the blending factors. We avoid storing floating point values by using one byte per entry. As a result, we have 255 morphing steps, which is precise enough in practice.

2.4 Clipping

A common improvement to reduce the number of polygons to be rendered is clipping against the viewing frustum. As we are already using a quadtree, we can also use it for clipping. Provided, the level of detail is not too high, a rectangular bounding box is computed for each node, which is used for clipping against the viewing frustum. In this way, most invisible vertices can be discarded at little cost at an early stage of the algorithm. Clipping can be applied both to the mesh generation and rendering phase. For mesh generation we consider bounding boxes to be three times as large, because the blending factors of some blocks can contribute to mesh generation without the blocks being visible themselves.

3 Results

All screen shots shown here have been taken from the application running on a SGI Maximum Impact workstation with a 250 MHz R4400 processor, 2 raster manager boards, and texture memory extension (TRAM option card), while maintaining a constant frame rate of 25 Hertz.

Images 9 and 10 show the height field and texture map used in the following examples. The data describes a region south of Haines Crossing in Yukon Territory, Canada.

In the Images 11 and 12 the point of view used for generating the triangulation has been located on the landscape's surface in the center of the image. The levels of detail clearly depend on both distance to the point of view and on surface roughness. The quality control C was set to a value of 8, which also proved to be a good choice for other data sets. The value of c was dynamically chosen as to maintain a fixed frame rate of 25 Hertz, which resulted in approximately 1600 triangle fans, or a total of roughly 15000 vertices per frame. This corresponds to a two orders of magnitude reduction of the original 1025×1025 height field.

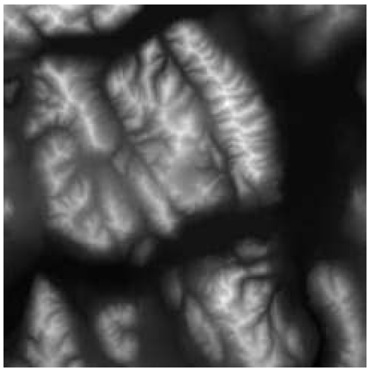

Figure 9: Height field used for examples.

Figure 10: Texture Map used for examples.

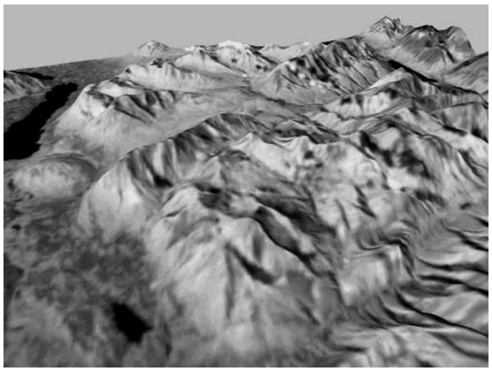

Figure 11: Rendered Yukon landscape.

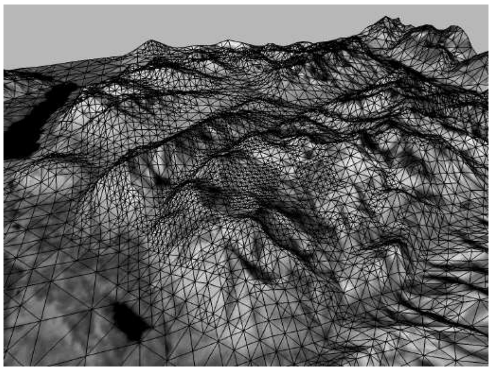

Figure 12: Superimposed triangulation.

Figure 13: Difference image of full and reduced resolution.

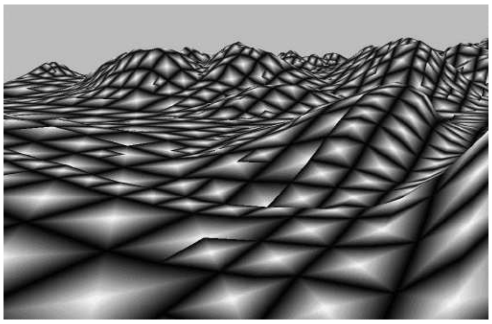

Figure 14: Example showing a typical triangulation generated by our algorithm.

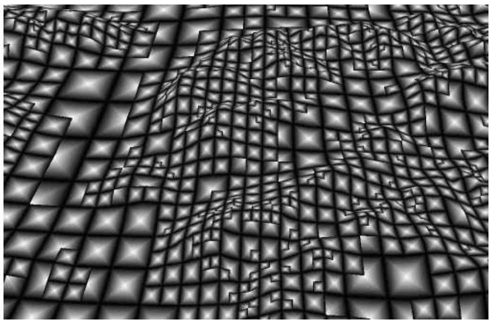

Figure 15: Example showing a typical top view triangulation.

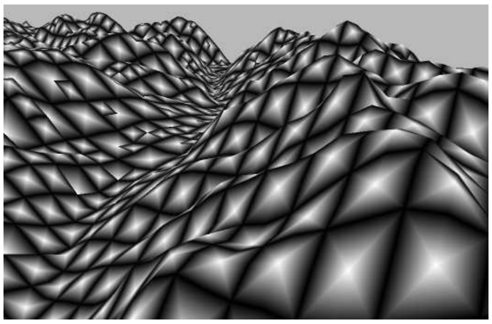

Figure 16: A typical valley view.

Image 13 shows the difference image between a real-time screen shot (Image 20) and rendering the complete height field (Image 21). The image quality is worst at the silhouettes. The human eye, however, is more sensitive to sudden changes of geometry, which have been significantly reduced by the geomorphing algorithm.

Images 14, 15 and 16 show some sample triangulations generated by our algorithm.

Images 17, 18, and 19 illustrate how triangulation accuracy changes with varying frame rate.

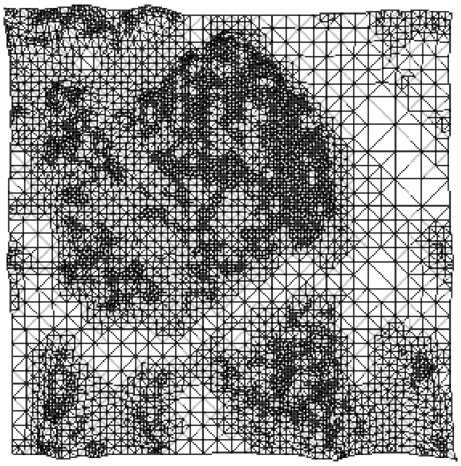

Figure 17: Triangulation for maintaining a frame rate of 25 Hertz.

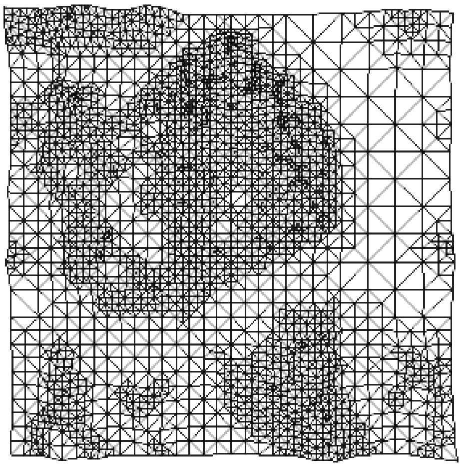

Figure 18: Triangulation for maintaining a frame rate of 38 Hertz.

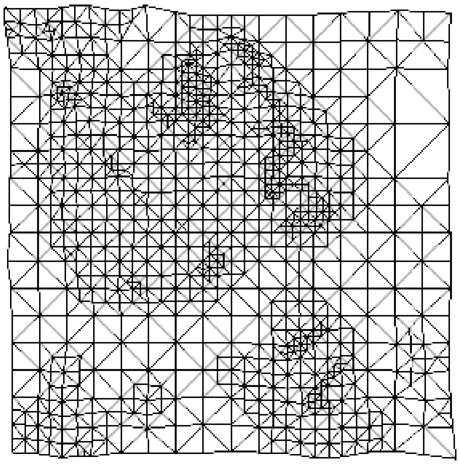

Figure 19: Triangulation for maintaining a frame rate of 76 Hertz.

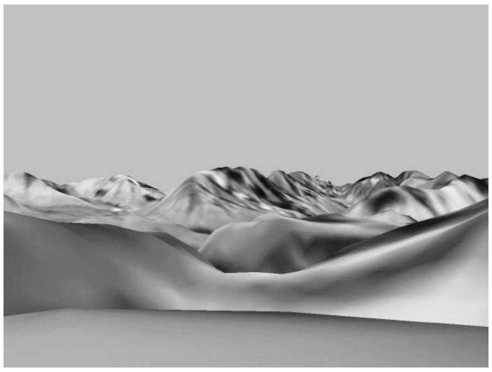

Figure 20: Triangulation for maintaining a frame rate of 25 Hertz.

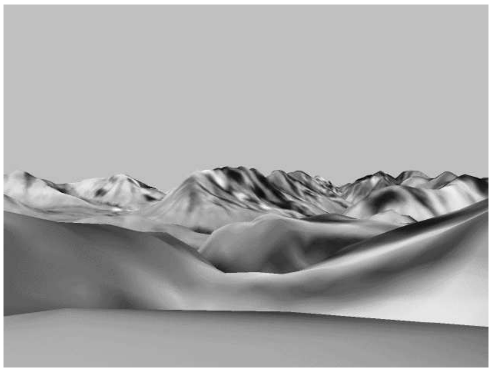

Figure 21: Same view as Figure 20 but with full resolution.

The memory consumption of the final algorithm is fairly low. Besides height field and texture map data only the $d2$-values and blending factors have to be stored. If $d2$-values are compressed to byte format, which can be done in a linear or nonlinear way, there is place enough in the quadtree matrix to store those values as well. In the end, we get away with only a single additional byte per grid point.

We are currently working on an efficient paging mechanism, that allows to render height fields that do not entirely fit into RAM.

4 Acknowledgments

We would like to thank Peter Lindstrom from Georgia Tech for valuable hints and discussions.

References

[Boa92] OpenGL Architecture Review Board. *OpenGL Reference Manual.* Addison-Wesley, 1992.

[GGS95] M. H. Gross, R. Gatti, and O. Staadt. Fast multiresolution surface meshing.

In G. Nielson and D. Silver, editors, *Proceedings Visualization '95*, pages 135–142. IEEE Computer Society Press, 1995.

[Hop96] Hugues Hoppe. Progressive meshes. In *Computer Graphics (Proceedings of Siggraph '96)*, pages 99–108, 1996.

[Hop97] Hugues Hoppe. View-dependent refinement of progressive meshes. In *Computer Graphics (Proceedings of Siggraph '97)*, pages 189–198, 1997.

[KLR+96] David Koller, Peter Lindstrom, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory Turner. Virtual GIS: A real-time 3D geographic information system. In G. Nielson and D. Silver, editors, *Proceedings Visualization '95*, pages 94–100. IEEE Computer Society Press, 1996.

[LKR+96] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory Turner. Real-time, continuous level of detail rendering of height fields. In *Computer Graphics (Proceedings Siggraph '96)*, pages 109–118, 1996.

[NDW93] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide.* Addison-Wesley, 1993.

[SN95] M. Suter and D. Nüesch. Automated generation of visual simulation databases using remote sensing and GIS. In G. Nielson and D. Silver, editors, *Proceedings Visualization '95*, pages 135–142. IEEE Computer Society Press, 1995.

[TB94] David C. Taylor and William A. Barrett. An algorithm for continuous resolution polygonalizations of a discrete surface. In *Proceedings of Graphics Interface '94*, pages 33–42, 1994.