# Multiresolution based on View-Dependent Progressive Meshes

Markus Grabner
grabner@icg.tu-graz.ac.at

Computer Graphics and Vision
Graz University of Technology
Graz / Austria

**Abstract**

Despite technological progress the increasing user demands require sophisticated algorithms for rendering and transmitting large 3D models. The key to solving these problems is *multiresolution*. This article shortly discusses some multiresolution methods developed during the past decade. The recent *progressive meshes (PM)* approach got much attention because it is very suitable for these requirements. The most important ideas regarding the PM framework are presented. Finally an implementation based on the PM representation is discussed including some modifications and extensions.

## 1 Introduction

During the last years computers are more and more used for visualization of different kind of data. Due to the fast technological progress it is now possible to handle large data sets. Viewing and manipulating of CAD models, medical data created by computer tomography, or digital elevation models constructed from satellite images is no longer restricted to supercomputers. Although mid-range graphic workstations can do all these things, there is the law of nature that computers are too slow. In the same amount as computational speed is increasing, the user's demands are also increasing (or even more). Some performance problems can (only) be solved by waiting six months for the next computer generation. But intelligent algorithms for the rendering of triangle meshes can achieve on today's hardware a performance otherwise not to be expected within the next years. This article explains a method called *Progressive Meshes* introduced by HUGUES HOPPE in [5] and extended in [6, 8].

One common approach to speed up graphics output is *multiresolution*, all objects are availabe at different resolutions. Among these resolutions the coarsest one sufficient to meet some user-defined fidelity goal is actually used for rendering. One can not expect to gain as dramatic improvements as with other problems (e.g. using binary search with $T(n) = O(\log n)$ instead of trivial search with $T(n) = O(n)$). Nevertheless the number of primitives required for an adequate approximation is usually much less than the total number of primitives for the whole model. It is desirable to have an *output-sensitive* algorithm, i.e. one with a rendering time of

$T(n, m) = O(m)$, where $n$ is the size of the data set, and $m$ is the area on screen covered by the projection of the object. An algorithm with this property can speed up rendering by two ways:

- Small objects that are far enough away from the camera occupy only a small fraction of the display area. They can be approximated with few primitives without visual loss of accuracy.
- Large objects that are only partly visible don't need to be sent to the graphics pipeline as a whole. Those parts lying outside the *view volume*[1] are clipped.

Note that reducing the number of triangles not necessarily reduces the number of pixels drawn. Instead, the geometry subsystem (see [3]) gets less work to do preventing it from being the performance bottleneck. Progressive Meshes as described in [6] support these features and are explained in more detail in section 3.

# 2 Survey on multiresolution methods

There is a large number of different multiresolution approaches, each one with its preferred application. A more extensive summary can be found in [13].

The methods differ greatly in the time required for preprocessing. Multiresolution models are usually built iteratively by applying a sequence of simplification steps. Naturally, the more time is spent for generating the multiresolution representation, the better is the approximation of the original data at different resolutions. Selecting the optimal sequence for simplification can significantly reduce the number of triangles required to achieve a fixed approximation quality. Typically methods that take into account the original mesh at each simplification step produce better results at the cost of a more sophisticated (and time consuming) algorithm. This is necessary, if a *global* constraint is given by the user (although it is possible to take advantage of the locality of the individual steps, as used e.g. in [9, 11]). This constraint can be expressed in different ways. Common choices include defining a geometrical error metric or using an energy function (as in [9]) to measure the accuracy of an approximation. On the other hand, using a *local* criterion for the guidance of the simplification process can be much faster. In this case, however, it is hard to assure a bounded precision, because locally estimated errors need to be accumulated in some way making an exact measurement impossible.

It is not always clear to which group a particular algorithm belongs. Some method of one class might be considered a special case of another class. The following list should be understood as a rough grouping of different methods according to their most important properties.

[1] the volume whose projection fits on the screen, usually a frustum or a quader, see [3]

## 2.1 Level Of Detail (LOD) switching

The simplest approach to multiresolution is explicitly providing the same object at different resolutions. Each mesh is stored independently, therefore only few levels of detail can be used. For this reason switching between two different representations causes distracting popping effects. Furthermore one single object can't be selectively refined. Viewing a large terrain model requires the whole mesh (even those parts far away from the camera) to be rendered at the same resolution that is necessary for a sufficient approximation near the viewpoint. Splitting a large object into several smaller parts is also a bad idea, although this can reduce the rendering time in some cases. But the borders between two patches at different resolutions become clearly visible, even cracks in the surface can occur.

## 2.2 Decimation

These methods iteratively remove elements (vertices, edges or triangles) from the mesh. When removing a triangle, it is replaced by a single vertex that is chosen to minimize some error metric. After updating the neighborhood this step is repeated until some stop criterion is reached. Other approaches remove vertices and retriangulate the resulting holes, thus successively simplifying the mesh. The removal of edges and their adjacent faces is also possible. As stated above, Progressive Meshes can be considered a special case of edge decimation.

Different proposals were made to bound the error of the simplified mesh. Some methods define an error volume surrounding the original mesh. Simplification stops when the simplified mesh is no longer completely contained in this error volume. In [11] the HAUSDORFF-distance is explicitly measured between regions of topological correspondence. The *deviation spaces* explained in section 4.1 are constructed in a similar way.

Another method worth mentioning is presented by LINDSTROM et al. in [12]. It is used for (and restricted to) the rendering of height fields. The mesh must be defined over a regular grid. Adjacent triangles are merged when the screen-space geometric error introduced by this operation is below a user-definable threshold. This error is measured as the projection to the screen of the vertical deviation between the simplified and the original mesh. HOPPE refers to this idea in [6].

## 2.3 Clustering

Clustering methods are not restricted to a particular topological type of their input and output data. In the simplest and fastest case vertices are clustered by discrete gridding and coordinate truncation. All vertices in one cluster are replaced by a single vertex. Edges and triangles that have become degenerate are removed. This procedure results in a simpler mesh, which might have a different topological type.

An interesting approach is described in [4]. It combines techniques used for Progressive Meshes and clustering methods. The authors report very good results for objects consisting of many separate parts.

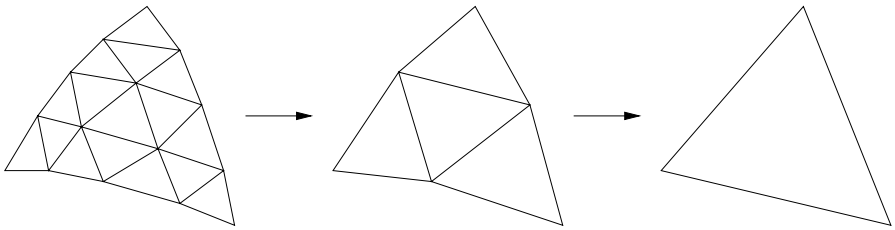


Figure 1: Simplifying a mesh with subdivision connectivity

### 2.4 Wavelets

Wavelet-based algorithms do not explicitly manipulate the elements of the input mesh. Instead, its geometry is represented by *multiresolution analysis* in a more abstract way. You can think of this process as applying a low pass filter to the mesh creating a coarser mesh. At the same time, an high pass filter is used to capture the details (stored as *wavelets*) that are lost due to low pass filtering. This procedure can only be performed on meshes with subdivision connectivity (see figure 1). For more details on wavelets see [1].

Only few meshes in practice satisfy the subdivision connectivity constraint. Therefore methods are required to transform arbitrary meshes to this special type. In [2] a solution is presented based on a discrete VORONOI-diagram. It is built upon the input mesh to separate it into triangular patches which are then *remeshed* with subdivision connectivity.

## 3 Review of Progressive Meshes

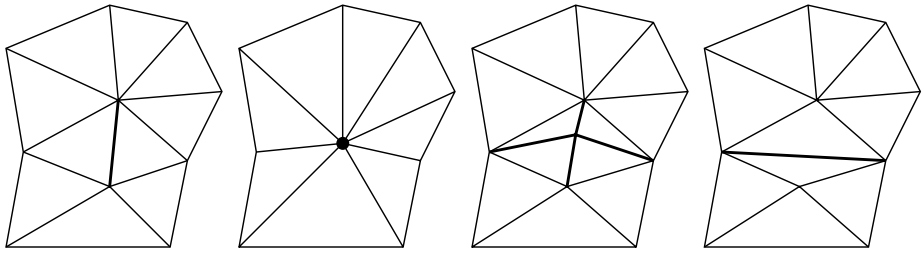
This section summarizes the ideas presented by HOPPE in his recent work on Progressive Meshes. The results of [7] are not discussed here because they are optimized for the non-viewdependent case.

### 3.1 Mesh Optimization

Although the approach in [9] is not immediately related to multiresolution, it introduces some concepts HOPPE'S further work is based on. The primary goal of this paper is to approximate surfaces reconstructed from unorganized points. Therefore the input data is assumed to be a set of points rather than a continuous surface.

The tradeoff between the number of vertices and the approximation quality is defined by an energy function. Thus large geometric errors are penalized as well as a large number of vertices, where the relative weight between these contradictory goals is defined by the user. The optimization procedure tries to find the mesh with minimum energy.

Optimization is performed by two nested loops. The outer loop modifies the mesh connectivity by randomly selecting an edge and performing one of three pos-



(a) initial patch (b) edge collapse (c) edge split (d) edge swap

Figure 2: Local operations to modify the mesh connectivity

sible edge operations (collapse, split, and swap, see figure 2) on the associated surface patch. Operations modifying the topology are disallowed. The inner loop chooses the vertex locations for fixed connectivity to minimize the energy function discussed above. Locality is exploited by only considering vertices in the neighborhood of the modified edge. The optimization procedure stops if no more edge operations are found that could further reduce the energy.

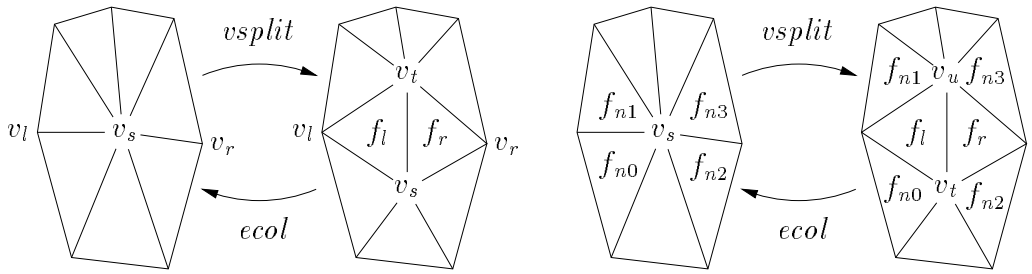
## 3.2 Progressive Meshes

The results explained in the previous section are extended in [5] to define a multiresolution representation called *Progressive Meshes (PM)*. It is constructed by iteratively applying the edge collapse operation to the input mesh until some error threshold or a specified number of faces is reached. The edges are no longer selected at random. Instead they are taken from the head of a priority queue sorted by increasing error introduced by the collapsing operation. Therefore the mesh is a good approximation at each step (not only at the end of simplification). When no more edge collapse operations are performed, the resulting mesh (the *base mesh*) is stored together with a sequence of *split records* as the PM representation. The *vertex split* operations are the inverse of the corresponding edge collapse operations (see figure 3(a)).

The PM representation is also able to take into account attributes such as color and normal vectors. Furthermore discontinuity curves defined as abrupt changes in these attributes between adjacent faces are considered. Avoiding modifications of the topology of discontinuity curves improves the appearance of the simplified mesh. The energy function (see section 3.1) is extended two deal with these two concerns.

## 3.3 View-Dependent Refinement

View-Dependent Refinement adresses the problems described in section 2.1. Although it was already considered in [5], it turned out that some modifications of the original approach are required ([6]). View-dependence is integrated into the PM




(a) according to [5]

(b) according to [6]

Figure 3: Different definitions of the mesh operations *vsplit* and *ecol*

framework by only performing those vertex split operations required for a certain screen-space error. Because the mesh operations can't be performed independently, a PM hierarchy tree is to be considered instead of a simple list. It proved useful to redefine the ecol/vsplit operations as shown in figure 3(b). The vertex split is legal if the faces $f_{n0}$ to $f_{n3}$ are already present in the mesh.

Some conditions have to be tested for each vertex to decide if it should be split or not. In [6], viewing volume, backfacing, and screen-space error tests are used. A vertex obviously doesn't need to be split if it lies outside the viewing volume or belongs to a backfacing region. If the vertex is visible from the current viewpoint, an estimation of the screen-space error introduced by the corresponding edge collapse is needed. HOPPE extends the metric used in [12] and defines an error volume called *deviation space* (figure 4(a)), whose projection to screen is used for the screen-space error test. All these conditions are packed into a single function qrefine which returns *true* if a vertex should be split and *false* otherwise.

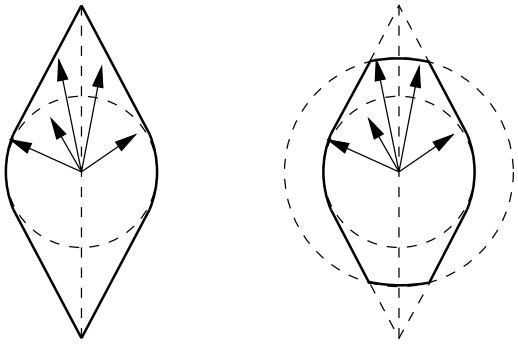
## 3.4 Smooth LOD Control

In [8] view-dependent PMs are applied to the rendering of very large terrain models. Memory usage is minimized through the use of output-sensitive data structures. Moreover, the PM representation is stored in a way that it doesn't need to be kept entirely in main memory.

Geomorphing[2] between two static meshes was already possible in [5, 6]. Now a scheme is presented for real-time applications. While the mesh connectivity is modified immediately when a vertex split becomes necessary, the associated geometric changes are distributed over several frames. This technique is simple to use for the (very likely) case of forward movement of the viewer. However, it has some drawbacks, especially when the viewer moves backwards. In section 4.2 an alternative approach is presented.

HOPPE also describes a procedure that alleviates memory problemes during preprocessing. The mesh is recursively split into smaller blocks which are then

[2] smooth geometric interpolation between two objects



(a) HOPPE ([6]) (b) extended

Figure 4: Approximation of the error volume by different deviation spaces

simplified independently up to a certain error threshold. After that the individual blocks are stitched together, and simplification is continued with the new larger blocks. This strategy enables preprocessing of huge models even on workstations with moderate main memory sizes.

# 4 Implementation

The implementation is mainly based on [6]. This section explains some modifications and extensions.

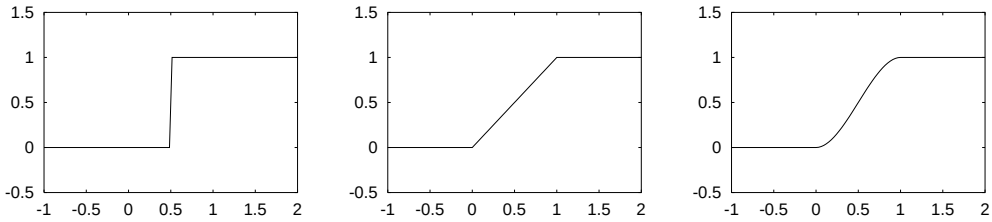
## 4.1 Deviation spaces

A deviation space as used in [6] is shown in figure 4(a). The error vectors (representing the geometric deviation between the original and the simplified mesh) are bounded by a volume consisting of a sphere and two cones. In this implementation the deviation spaces are "clipped" at a sphere large enough to contain all error vectors (figure 4(b)). Thus the tops of the cones are cut off resulting in less detail drawn without exceeding the screen-space error tolerance.

After a vertex $v_s$ is created by an edge collapse operation, for all leaf nodes of the hierarchy tree below $v_s$ (vertices of the original mesh) their nearest neighbor on $N_{v_s}$ (all faces adjacent to $v_s$) is computed. The error vectors are the deviations between a leaf vertex and its nearest neighbor on $N_{v_s}$. By descending the PM hierarchy it is also guaranteed that only topological corresponding regions are considered (as in [11]).

## 4.2 Geomorphing

HOPPE suggests that geomorphing sequences should cover a constant time, typically one second. This causes two problems. After an immediate stop in the viewer's




(a) boolean (b) linear (c) smooth

Figure 5: Modifying the function qrefine to support geomorphing

movement the mesh needs this constant time for stabilization. Moreover, it is hard to deal with the PM dependencies when geomorphing is used to smoothly coarsen the mesh.

An alternative approach not explicitly dealing with time redefines the function qrefine (see section 3.3) to have continuous range. Figure 5 shows the output of different types of this function for a visible vertex (inside the viewing-volume, not backfacing). The $x$-axis is the screen-space geometric error (the projected size of the deviation space), using an arbitrary scaling for demonstration purposes. With qrefine defined as in figure 5(b) or 5(c) flexible geomorphing sequences are generated depending on the current viewing parameters instead of time. Such a sequence can even be reverted before it is finished.

For reasons not explained here a more sophisticated algorithm is required for the traversal of vertices to be evaluated with qrefine. However, this procedure can be optimized, so only little computational overhead is added.

## 4.3 Parallel preprocessing

Meshes worth being simplified with any multiresolution method usually consist of many thousands of triangles. Therefore it is very unlikely that two consecutive edge collapse operations affect neighboring regions. A parallel preprocessing algorithm can take advantage of this property. This implementation is based on POSIX threads. Several concurrent threads try to perform the local mesh operations (edge collapses) in parallel. Before modifying the mesh it must obviously be guaranteed that no other thread is currently working on the involved region (marked red in figure 6). Note that the red regions are directly modified by collapsing the thick edges and therefore must not overlap. The green regions (consisting of all faces containing at least one green vertex) are also required to be untouched during the creation of the simplification record. But because only read accesses are performed, they may safely overlap as indicated in figure 6. A proper locking technique applied to the green vertices is sufficient to avoid conflicts.

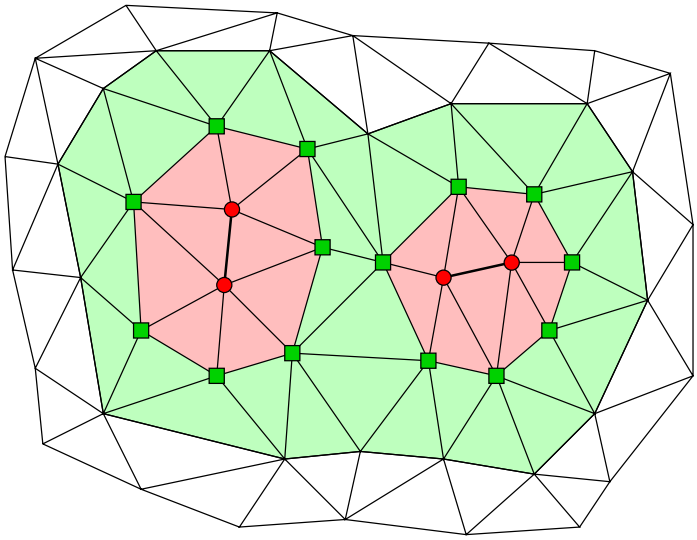


Figure 6: Locking considerations for parallel preprocessing

### 4.4 Progressive Transmission

As already indicated in [2, 5], an important application for multiresolution models is *progressive transmission*. Transmission of the whole model might take minutes or even hours for a large mesh using a low-bandwidth network. Therefore a method is required to successively transmit the surface geometry starting with a coarse simplification. The PM approach supports progressive transmission in a quite natural way because the data is already stored as a coarse base mesh and a sequence of detail records. Transmitting the whole object while getting an early impression of its appearance is easily and efficiently done by transmitting the PM representation.

In the view-dependent framework the client only requests a small part of the object stored at the server. In this implementation a simple approach was chosen where the client requests any information not yet received. Alternatively the client could send its viewing parameters to the server, which can then determine the data needed by the client without further communication. However, the latter method has disadvantages during rapid movement.

## 5 Results

Unfortunately the implementation has not yet reached a point where reliable run times and similar statistics can be reported. Instead some qualitative observations are presented. Some images (figure 7) further document the results.

The deviation spaces (section 4.1) succeed in adding detail near the silhouettes as reported in [6]. While this works well for regions with low curvature, the deviation spaces degenerate to spheres after few simplification levels for rough regions. This is not surprising, but an algorithm that delays the degeneration to higher levels as far as possible would be desirable.

Geomorphing as described in section 4.2 produces very attractive results. The only drawback of the static approach seems to be the presence of many nearly degenerate triangles under certain circumstances. However, this is only visible in wireframe mode because texture coordinates and normal vectors are interpolated together with the vertex locations.

Parallel preprocessing was tested on a Dual-Pentium-II board. A rough estimation of run times revealed that about 8% of the simplification code run in critical sections (i.e. only one thread at a time). That results in a lower bound of 54% of execution time compared to a uniprocessor machine. In experiments the execution time could be reduced to 64%. While this result is not too bad for a two-processor machine, some improvements might increase parallelism. At the moment the priority queue seems to be the bottleneck when using the concurrency scheme of section 4.3.

No detailed statistics are available about progressive transmission (section 4.4) by now. First experiments indicate that transmission takes place in reasonable times even at a low bandwidth. An integrated utility allows the bandwidth to be explicitly limited so that the effect of using modem lines and other typical transmission media can be explored.

# 6 Summary and future work

After giving a short survey on recent multiresolution methods, the progressive meshes representation was presented. This is a promising multiresolution approach that can be extended to support view-dependent refinement and real-time applications. Future work includes extending the refinement criteria to account for surface shading as it was already done in [14, 10].

The implementation described in section 4 is the current state of my diploma thesis. I hope that the reader is now curious about future results in this young and dynamic field of research and might even be interested in the final version of this work. The diploma thesis is expected to be finished in 1999 and will reveal many details which are beyond the scope of this article.

# References


[1] Michael F. Cohen, Tony D. DeRose, Alain Fournier, Leena-Maija Reissell, and Peter Schröder. *Wavelets and their Applications in Computer Graphics.* SIGGRAPH '94 Course Notes. 1994.

[2] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 173–182. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[3] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition.* Addison-Wesley, Reading, Massachusetts, 1990. Overview of research to date.

[4] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[5] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[6] Hugues Hoppe. View-dependent refinement of progressive meshes. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[7] Hugues Hoppe. Efficient implementation of progressive meshes. Technical Report MSR-TR-98-02, Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, January 1998.

[8] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization '98*, pages 35–42, October 1998.

[9] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, August 1993.

[10] Reinhard Klein. Multiresolution representations for surfaces meshes. Technical report, Wilhelm-Schickard-Institut, GRIS, Universität Tübingen, Germany, June 1997.

[11] Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. Mesh reduction with error control. In *IEEE Visualization '96*. IEEE, October 1996. ISBN 0-89791-864-9.

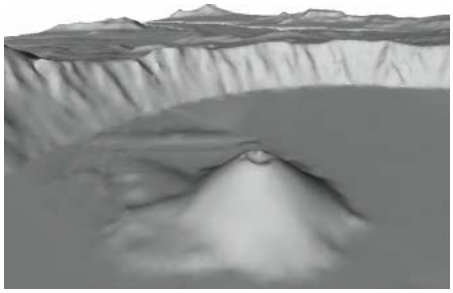
[12] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 109–118, 1996.

[13] Enrico Puppo and Roberto Scopigno. *Simplification, LOD and Multiresolution, Principles and Applications.* Eurographics Tutorial. Blackwell Publishers, 1997.

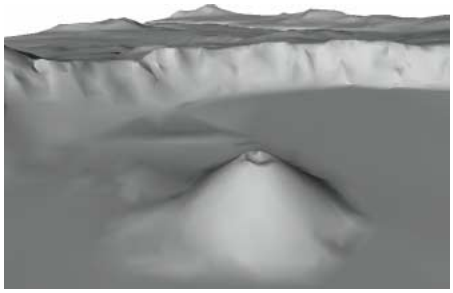
[14] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96*. IEEE, October 1996. ISBN 0-89791-864-9.

(a) original bunny (69451 triangles)

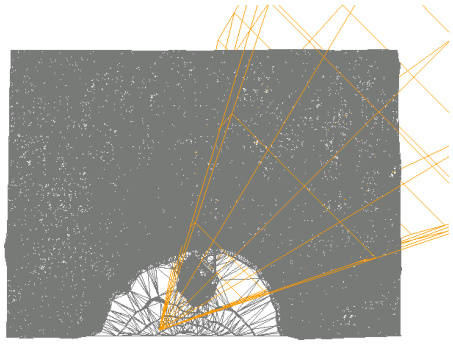
(b) simplified version (2998 triangles)



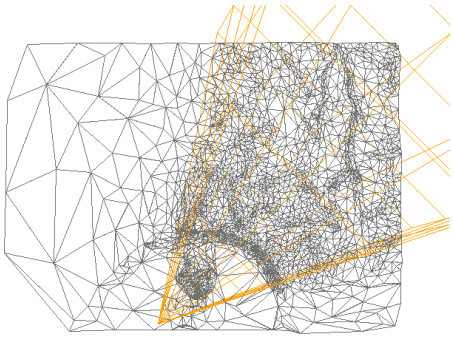
(c) original terrain model (199114 triangles)



(d) simplified using view-dependent refinement (4995 triangles)



(e) original model in wire-frame representation



(f) simplified model with viewing parameters of figure 7(d)

Figure 7: Images generated with the implementation discussed in section 4