Efficient Adaptive Simplification of Massive Meshes

Eric Shaffer* Michael Garland

University of Illinois at Urbana–Champaign

Abstract

The growing availability of massive polygonal models, and the inability of most existing visualization tools to work with such data, has created a pressing need for memory efficient methods capable of simplifying very large meshes. In this paper, we present a method for performing adaptive simplification of polygonal meshes that are too large to fit in-core.

Our algorithm performs two passes over an input mesh. In the first pass, the model is quantized using a uniform grid, and surface information is accumulated in the form of quadrics and dual quadrics. This sampling is then used to construct a BSP-Tree in which the partitioning planes are determined by the dual quadrics. In the final pass, the original vertices are clustered using the BSP-Tree, yielding an adaptive approximation of the original mesh. The BSP-Tree describes a natural simplification hierarchy, making it possible to generate a progressive transmission and construct level-of-detail representations. In this way, the algorithm provides some of the features associated with more expensive edge contraction methods while maintaining greater computational efficiency. In addition to performing adaptive simplification, our algorithm exhibits output-sensitive memory requirements and allows fine control over the size of the simplified mesh.

Keywords: surface simplification, massive meshes, quadric error metric, recursive partitioning, out-of-core simplification

1 INTRODUCTION

Recent advances in three-dimensional model acquisition and tertiary storage technology have resulted in the wide availability of massive geometric data sets. The Digital Michelangelo Project [15] at Stanford University has produced finely detailed polygonal meshes containing up to two billion triangles. In the scientific domain, the Visible Human Project has generated data sets of over 10 billion voxels, while the DOE ASCI project has a goal of visualizing multi-gigabyte iso-surface datasets.

The enormity of these models poses serious challenges for visualization. Current workstation technology is completely incapable of rendering them in real-time. In the case of polygonal meshes, applying a simplification algorithm would seem to be a reasonable approach. Unfortunately, most traditional simplification algorithms have memory and processing time requirements that are far too high to handle massive data. Relatively few can process million-face models, and meshes with hundreds of millions of faces are far beyond their capacity. Moreover, many of these algorithms require random access to the mesh data, and so cannot be easily adapted to work efficiently on meshes exceeding the size of main memory. As a result, these algorithms perform poorly, or not at all, when applied to the very meshes most in need of simplification. To date, only uniform clustering has been successfully applied to truly massive models. However, the approximations generated with this method can display artifacts related to the regularity of the sampling grid.

In order to work efficiently with very large data, we believe an algorithm must obey several key constraints. Since the magnitude of the input data will exceed memory capacity, it is critical that the size of in-core data structures be independent of the input size. Expensive operations, such as simplification, must be performed only on data held in-core. Since the input data is kept out-of-core, access must be limited to a small number of linear scans in order to maintain efficiency. Adhering to these principles, we have designed a memory efficient algorithm for adaptive simplification of massive polygonal meshes. While it maintains computational efficiency close to that of uniform clustering, our method can yield higher quality approximations. The algorithm can be thought of as a three step process which begins by quantizing the input mesh using a uniform grid. This grid is used to accumulate surface information in the form of quadrics and dual quadrics, which are used to characterize the local shape of the surface. In the second step, this geometric information is used to build a data structure that effectively partitions the vertices of the model. Finally, the original mesh is simplified using this partition information.

Employing an effective vertex partitioning method is critical to both the performance of the algorithm and quality of the simplified mesh. Inspired by the R-Simp method of Brodsky and Watson [3], we have developed a spatial clustering technique based on BSP-Trees [24]. This choice has several appealing properties. Simplification proceeds from a coarse to fine level, allowing us to efficiently generate very coarse approximations of large meshes. Our BSP structure is adaptive, with the planes chosen according to heuristics based on the local geometry of the mesh. Also, the natural hierarchy provided by the BSP-Tree makes it particularly suited for applications such as progressive transmission and constructing level-of-detail hierarchies. While R-Simp shares these characteristics, it can only operate on meshes small enough to fit in-core. By employing a quantization phase and new heuristics for constructing the BSP-Tree, the algorithm we present is capable of simplifying models orders of magnitude larger than those suitable for R-Simp.

The primary contribution of the work described in this paper is a novel algorithm for efficient, adaptive simplification of massive meshes. This simplification algorithm is memory efficient, consuming memory in proportion to the size of the output mesh. It is also time efficient, yielding running times that are competitive with uniform simplification. Since only two linear scans are made over the input data, the algorithm can scale to handle massive datasets. Finally, our algorithm is, to our knowledge, one of the few methods able to perform adaptive surface simplification of a massive mesh. It may be the only method able to do so efficiently.

2 PREVIOUS WORK

Polygonal simplification [7, 4] has been an area of active research for close to a decade. Optimal approximation of a surface is known to be NP-Hard [1], and hence most research has focused on developing heuristic methods. The fundamental principle underlying most of these methods is that of partitioning the vertex set of a mesh

*Department of Computer Science, University of Illinois, Urbana, IL 61801. shaffer1@cs.uiuc.edu, garland@cs.uiuc.edu

into disjoint clusters and unifying the vertices within each cluster. This process of vertex unification will cause some faces in the mesh to degenerate into edges or points. The end result is a polygonal surface approximation with fewer vertices and faces. Simplification methods are generally differentiated based on how they construct the partition of the vertex set. One of the more common approaches is the application of iterative edge contraction.

Iterative edge contraction has become arguably the most popular heuristic framework. Techniques based on this operation select a pair of vertices at each iteration and replace them with a single vertex, removing any triangles that degenerate. This process effectively partitions the original vertex set, as each vertex in the approximation represents a set of vertices that have been contracted together. Edge selection is generally accomplished through a greedy strategy, with the cost of an edge contraction being related to the error it induces in the approximation. The quadric error metric, introduced by Garland and Heckbert [9], has proven to yield good results in practice.

Unfortunately, edge contraction algorithms are not well-suited to work on massive meshes. They typically employ a priority queue of possible contractions, resulting in memory consumption proportional to the size of the original mesh. This is clearly untenable for extremely large models. Even if this obstacle is overcome by using out-of-core data structures, the order of contraction operations will often exhibit little locality, meaning each contraction will be expensive. Given the large number of contractions required to satisfactorily simplify a massive model, the cost of such a computation is prohibitive. Rather than resorting to out-of-core data structures, several authors, Bernardini [2], Hoppe [13] and Prince [21] in particular, have proposed methods in which a mesh is segmented so that each piece fits in main memory. The pieces are then simplified in-core, with the boundary edges preserved so the segments can be rejoined. This process is iterated, with new boundary edges chosen each time. While this solution is conceptually appealing, the segmenting and rejoining operations are expensive, making this approach less attractive for very large meshes. El-Sana and Chiang describe a somewhat similar method [6] for out-of-core view-dependent refinement. Initial simplification is accomplished using an out-of-core priority queue in combination with segmentation. The result is an out-of-core structure that supports rendering of large meshes using levels-of-detail. While the authors achieve interactive frame rates for some large models, the preprocessing time is significant.

Another popular simplification framework, and one more suited to large models, is spatial clustering. These methods spatially partition a vertex set into clusters and unify all vertices within a given cluster. Rossignac and Borrel developed one of the earliest clustering algorithms [22]. Their method partitions a vertex set using cells defined by a rectilinear grid. A representative vertex is synthesized for each cell. When triangles from the original mesh are mapped onto these new vertices, many faces degenerate into points or lines and are discarded, thereby simplifying the model. Low and Tan [19] proposed an adaptive variant of this technique, called floating-cell clustering, which ranks the vertices of a model by importance and iterates over the vertex set. In each iteration, a cell of user-specified size is centered on the most important vertex and all vertices falling within that cell are merged. The hierarchical dynamic simplification system of Luebke and Erikson [20] constructs a vertex tree, essentially a hierarchy of vertex clusters, that is used to provide view-dependent simplification at runtime.

Recently, Lindstrom [16, 17] extended the Rossignac-Borrel algorithm to work on models too big to be held in-core by formatting the input mesh as a polygon soup. Reading the model off disk in single linear scan, vertices are clustered and degenerate faces discarded. The quadric error metric is used to position the representative vertices, resulting in higher quality output than the original Rossignac and Borrel method.

While uniform clustering offers great efficiency, there are cases where the uniformity of the grid causes undesirable artifacts in the approximation. No feature on the input model smaller than a grid cell can be retained using uniform clustering. Also, large flat regions will be over-tessellated and the vertices in the output mesh will be distributed more or less evenly due to the structure inherent in the grid. Ideally, one would prefer to simplify using smaller polygons in areas with fine features and larger polygons elsewhere. The R-Simp algorithm of Brodsky and Watson [16] employs an adaptive vertex clustering technique to achieve this end. R-Simp builds a spatial decomposition by recursively splitting cells using the curvature of the model to determine the splitting planes. At each step, the cell exhibiting the greatest amount of area weighted curvature is split. This process continues, essentially proceeding from coarse to finer approximations, until the desired resolution is obtained. While this algorithm is more efficient than edge contraction methods, it cannot easily be used for processing large meshes. Computing the area weighted normal variation and finding a splitting plane for a cell both require an examination of all the mesh faces contained in the cell. Keeping the model in external memory, a necessity for massive data sets, would result in these operations being prohibitively expensive.

Rusinkiewicz and Levoy [23] describe an out-of-core multiresolution rendering system, QSplat, built around point-rendering rather than mesh simplification. QSplat uses a bounding sphere hierarchy to define an external memory representation for very large meshes. Rendering is accomplished by generating "splats" for each sphere in a cut through the hierarchy. While this system works quite well, there are applications where having an actual surface, such as a simplified mesh, is advantageous. Many existing rendering systems are geared toward polygonal data and can more readily work with a mesh based representation. Also, Hubbard [14] showed that bounding sphere hierarchies based on octrees do not generally fit surfaces well enough to provide accurate time-critical collision detection. We would expect the bounding boxes used to construct the QSplat hierarchy might suffer from the same problem. It is interesting to consider that the vertex clustering method we present could be used to build an adaptive bounding volume hierarchy suitable for the QSplat rendering engine.

3 QUADRIC QUANTIZATION

The initial step of our algorithm is to perform quadric quantization of the input mesh. This process begins by computing a dense partition of space and accumulating quadrics in the cells of the partition. We have chosen to implement this partition, like Lindstrom, as a uniform grid. The benefit of this approach is that it is possible to build such a grid without reference to the original mesh beyond knowing a bounding box. Touching the original data as little as possible is essential to maintaining efficiency when working with large data sets.

Our algorithm relies on the quadric and dual-quadric error metrics introduced by Garland and Heckbert [9, 10] to encode information about the geometry of a model in a memory efficient manner. Only ten coefficients are required to store a quadric, and the additive properties of quadrics enable information to accumulate within a grid cell without increasing the size of the data.

3.1 Quadric Metric

The primal quadric metric effectively measures the sum of squared distances from a vertex to a set of planes. Consider that each face in a mesh defines a plane which satisfies the equation $\mathbf{n}\mathbf{v}^{\mathsf{T}} + d = 0$, where $\mathbf{n}$ is a unit normal. The squared distance of a vertex $\mathbf{v}$ to this plane can be represented using a quadric Q:

(a) 28,184,526 vertices (Original)

(b) 107,872 vertices (0.3%) time=17:02

(c) 50,000 vertices (0.1%) time=16:10

(d) 5000 vertices (0.01%) time=13:48

(e) 1000 vertices (0.003%) time=12:21

Figure 1: Progressive approximations produced by our algorithm.

$$Q = (\mathbf{A}, \mathbf{b}, c) = (\mathbf{n}\mathbf{n}^\mathsf{T}, d\mathbf{n}, d^2)$$

$$Q(\mathbf{v}) = \mathbf{v}^\mathsf{T}\mathbf{A}\mathbf{v} + 2\mathbf{b}^\mathsf{T}\mathbf{v} + c$$

To compute the sum of squared distances to a set of planes, we only need a single quadric that is the sum of the quadrics defined by each of the individual planes.

In addition to its utility as an error metric, the quadric matrix also encodes information about the curvature of the associated set of planes. The 3 x 3 matrix $\mathbf{A}$ is the sample covariance matrix of the set of normals, with mean $[0, 0, 0]$. If the eigenvalues of the matrix are ordered from smallest to largest, the corresponding eigenvectors are the direction of minimum normal variation, maximum normal variation, and the average normal. For a smooth surface, the directions of minimum and maximum normal variation approximate the directions of minimum and maximum curvature [8].

3.2 Dual Quadric Metric

Just as the quadric metric encodes distance from a point to a set of planes, the dual quadric [8, 10] measures distance from a plane to a set of points.

Given a set of vertices $\{\mathbf{v}_1, ..., \mathbf{v}_k\}$, we define the dual quadric as

$$P_i = (\mathbf{D}_i, \mathbf{e}_i, f_i) = (\mathbf{v}_i{\mathbf{v}_i}^\mathsf{T}, \mathbf{v}_i, 1)$$

$$P_i(\mathbf{n}, d) = \mathbf{n}^\mathsf{T}\mathbf{D}_i\mathbf{n} + 2{\mathbf{e}_i}^\mathsf{T}(d\mathbf{n}) + f_i d^2$$

Given a set of vertices $\{\mathbf{v}_i\}$, summing the dual quadrics over all the vertices we have

$$P = (\mathbf{D}, \mathbf{e}, f) = (\sum \mathbf{D}_i, \sum \mathbf{e}_i, \sum f_i)$$

This allows us to express the covariance matrix for the set as

$$\mathbf{Z} = \mathbf{D} - \frac{\mathbf{e}\mathbf{e}^\mathsf{T}}{f}$$

This formulation neglects the $\frac{1}{k-1}$ averaging factor typically used in computing a covariance matrix, but the eigenvectors and the relative scale of the eigenvalues are unchanged by this. The eigenvector corresponding to the smallest eigenvalue is in fact the normal for the least squares best plane through $\{\mathbf{v}_i\}$. The eigenvector associated with the largest eigenvalue is the direction in which $\{\mathbf{v}_i\}$ exhibits the most spread.

3.3 Quantizing the Mesh

A quadric quantization is generated in a straight-forward manner. We simply scan a file describing the input mesh and hash the vertices of the mesh into a table describing a uniform spatial decomposition. In constructing this uniform grid, we assume the bounding box of the model is known. This is a modest requirement; most model acquisition methods can provide such information, and in any case, it is trivial to compute a bounding box by performing an extra linear scan of the mesh.

Each vertex in the mesh generates a dual quadric that is added to the containing grid cell. Keeping the standard assumption that faces are described as triangles, each face will generate a quadric which is then added to the three cells associated with the vertices of the face. In our current implementation, the input mesh is described as an indexed face set held in a file on disk and is accessed via memory-mapped I/O managed by the operating system. The need to look up the vertices corresponding to the face indices could be problematic if the face set is described without locality. In such a situation, vertex information would repeatedly be read from disk and thrashing would likely occur. Lindstrom's method is able to guarantee locality, at the expense of increased disk space, by representing the model as an unindexed polygon soup. While nothing prevents our algorithm from adopting a similar strategy, all of the models we have worked with so far have exhibited enough locality that generating a polygon soup representation has been unnecessary.

4 SIMPLIFICATION ALGORITHM

After quadric quantization of a model, our algorithm builds an adaptive data structure describing a spatial partition which is used to cluster the vertices of the model. Since working with the original mesh is virtually impossible to do efficiently, we rely instead on the quadric information gathered during the quantization step. The algorithm described by Lindstrom [16] uses such a grid to directly produce a simplified version of the model, with the size of the grid determining the size of the simplified mesh. In our algorithm, the grid is only employed as an intermediate approximation to the original mesh. This approximation is used to generate an adaptive spatial partition. Increasing the size of the uniform grid increases the quality of the approximation and resulting spatial partition, but does not directly affect the size of the simplified mesh. It should be

noted however, that the number of vertices in the simplified mesh is bounded above by the number of occupied cells in the uniform grid.

We chose to implement this spatial partition by constructing a BSP-Tree. Inspired by the R-Simp system of Brodsky and Watson, this approach has several appealing characteristics. It describes a natural simplification hierarchy, in which any cut through the tree describes a valid simplification. The generation of a such a hierarchy is a feature lacking in most spatial partitioning methods, but is generally provided by more computationally expensive simplification methods based on edge contraction. The hierarchy defined by the BSP-Tree enables applications such as progressive transmission and makes it easy to construct continuous levels of detail. Each interior node in the tree describes a vertex split from a single vertex into two vertices. This action is represented by the interior node being linked to two children. Although we have not implemented it, one could use the information provided by the the tree to perform vertex split and contraction operations within a view-dependent refinement system. Generating such a level-of-detail hierarchy is more difficult with uniform clustering, since the size of the simplified mesh is related to the grid resolution and difficult to control in an exact fashion. The BSP-Tree offers exact control over the size of output mesh; since each leaf generates a single vertex we simply split leaves until the desired number of vertices is reached. Figure 1 shows a set of progressively coarser approximations generated by our algorithm.

4.1 BSP-Tree Construction

Following quantization, we generate a representative point for each occupied cell in the quantization grid. The placement of this point is chosen to minimize the primal quadric error. These points are then inserted into a BSP-Tree. Each point carries with it a set of associated data, including the quadric and dual quadric from the grid cell it represents. The BSP-Tree initially has a single node containing all of the representative points.

We keep a priority queue containing the leaves of the BSP-Tree, keyed off of the primal quadric error for the leaf. Each leaf will likely contain many representative points, and the primal quadric for a leaf is simply the sum of the primal quadrics associated with those points. Similarly, the dual quadric for the leaf is the sum of the dual quadrics associated with the representative points. The construction of the tree proceeds as follows:

Until the desired number of leaves are created:

1. Choose the leaf with largest quadric error
2. Create a split plane using the dual quadric
3. Divide the leaf and enqueue the 2 new leaves

Each leaf in the tree will correspond to a vertex in the simplified model. By always splitting the leaf with the largest error, the algorithm introduces more vertices, and thus more detail, in areas that are poorly represented by a single vertex. The adaptivity that results from this process can be seen in Figure 2, showing the spatial partitions across the the surface of a model. Here, a sphere is positioned at the center of each vertex cluster generated by the BSP-Tree, with the size of the sphere related to the diameter of the cluster. Notice that the clusters are smaller and more numerous in areas of high-detail.

Our method for determining the splitting planes of the BSP-Tree is similar to the technique used by Gottschalk, Lin, and Manocha to build OBB-Tree bounding volumes [11]. Both rely mainly on point-set covariance, although the OBB-Tree computation involves sampling across the convex hull of the point set. Our algorithm uses the dual quadric associated with a leaf to determine a splitting

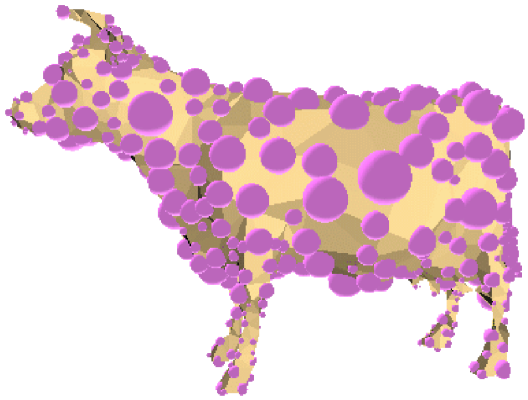


Figure 2: Adaptive vertex clustering on a polygonal mesh

plane. In the BSP-Tree, each leaf effectively specifies a space defined by a series of half-space intersections. Using the dual quadric, we are able to approximate the directions in which the vertex positions of the original mesh exhibit minimum and maximum spread.

Once we have determined which leaf to split, the eigenvectors and eigenvalues of the point set covariance matrix associated with that leaf are extracted. If the piece of mesh within the space corresponding to the leaf exhibits coherent point spread, it will likely look something like a rectangular patch. If we consider how to split this patch with a single plane to obtain a good approximation, it seems reasonable to choose a plane cutting perpendicular to the direction of greatest point spread. An edge connecting the two cells resulting from this split would run in the direction of maximum point spread, essentially using a line to approximate the patch lengthwise. In the event that the patch does not exhibit coherent point spread, it is possible that there are actually multiple surface sheets contained within the space associated with the leaf. In this case, we choose the normal of the splitting plane to be the direction of minimum point spread. This is done in hopes of keeping separate sheets from collapsing together.

These heuristics are implemented by examining the eigenvalues of the point set covariance matrix. If the ratio of largest to smallest is significant, the magnitude of variation is clearly greater in the direction of maximum spread and it is more likely the patch is coherent. In this case, the normal of the splitting plane is the direction of maximum spread. If, however, the ratio of the eigenvalues is less than 2, the surface information is considered to be incoherent and the normal of the splitting plane is chosen to be the direction of minimum point spread. In either case, we position the plane so that it passes through the mean vertex position, which can be extracted from the dual quadric.

4.2 Simplification

After the tree is constructed, we execute a second scan of the original model. For each face in the model, we must distribute the quadric to the leaves of the BSP-Tree containing the three vertices of the face. Each vertex is mapped to a leaf of the BSP-Tree by walking down the tree and testing the vertex against each plane on the path. If the three vertices fail to map to three different leaves, the face degenerates and is discarded. Once this scan is complete, a representative vertex position is computed for each leaf using its associated quadric. The simplified mesh consists of these vertex positions and the non-degenerate faces.

This second scan of the model is required because the vertex clusters, and hence the quadrics associated with each leaf, gener-

Grid dimensions	Memory (MB)	Error	Time (sec)
$15\times36\times15$	15	12.7×10^{-7}	9.0
$30\times72\times30$	25	9.3×10^{-7}	9.6
$60\times144\times60$	34	9.0×10^{-7}	10.0

Table 1: Simplification performance on the Stanford Buddha model as a function of grid resolution. The model was simplified from 543,652 to 812 vertices.

ated by the second pass will almost certainly differ from those computed during construction of the BSP-Tree. During the construction phase, the algorithm operates on representative points generated from the cells of the quantization grid. Each representative point has an associated quadric and dual quadric which are computed from the the vertices that fall into a particular grid cell. When a leaf is split during construction, these quadrics are added to one of the new leaf nodes resulting from the split. This leaf node is chosen based on the relationship of the representative point to the splitting plane. Unfortunately, not all of the vertices that generated the quadrics will necessarily lie on the same side of the plane. In many cases, the splitting plane might bisect the grid cell and the vertices will fall on both sides of the splitting plane. Re-clustering the original vertices, and generating new quadrics, removes this inaccuracy.

4.3 Memory Efficiency

The vast majority of memory use occurs in the construction of the quantization grid and the BSP-Tree. Since the size of each of these structures is essentially a function of the size of the simplified model, the algorithm maintains memory efficiency even when faced with very large input. The resolution of the quantization grid must be at least equal to the desired number of vertices in the simplified model. Since relatively few cells in the grid end up being occupied, the resolution will often need to be higher still, but it remains unrelated to the size of the original mesh. Implementing the grid as a hash table helps to limit the excess memory allocation required to generate a particular number of representative points.

Increasing the resolution of the grid, while holding the number of vertices in the simplified model fixed, decreases approximation error. This effect is illustrated in Table 1. Here, a model with 543,652 vertices and 1,087,716 faces is simplified down to 812 vertices. The increase in grid size results in greater memory use for both the hash table and the BSP-Tree, since more representative points will be held in the tree. A four-fold increase in the size of the grid in each dimension yields about a thirty percent reduction in error.

The amount of memory used by the BSP-Tree is directly related to the size and quality of the resulting approximation. Each vertex in the simplified mesh corresponds to a leaf in the tree. The total number of nodes in a tree with n leaves will be $2n-1$. In our current implementation, each node has a memory footprint of 160 bytes. In addition, each occupied cell in the quantization grid uses 144 bytes of storage. Since the size of the quantization grid is related to the desired quality of the approximation, and therefore is user-defined, it is difficult to derive a formula for theoretical memory usage. In comparing our method to that of Lindstrom, we can say that we will use several times the memory required by his algorithm. In generating approximations with the same number of vertices, our algorithm must use a quantization grid larger than the grid employed by uniform clustering in order to generate a better quality approximation. In addition, our grid cells will store twice as much quadric information. However, since both the grid and tree resolution are determined by the desired quality and size of the simplified mesh, memory consumption for our algorithm remains completely independent of input size. Moreover, adjusting the quantization grid allows one to affect the error of the approximation without changing the approximation size. Uniform clustering, in which changing the grid size changes the size of the simplified mesh, lacks this feature.

4.4 Discussion

While our method was inspired by the R-Simp system of Brodsky and Watson, there are key differences. Most significantly, by employing an initial quantization step, our algorithm is able to operate on meshes too large to fit in-core, whereas R-Simp is confined to operating on meshes small enough to fit in memory. The algorithms also differ in how they construct the adaptive spatial subdivision used to simplify the input mesh. Our determination of which leaf in the tree to split uses a different metric, as does our construction of splitting planes. R-Simp splits the cell in which the face normals exhibit the most variation, while we choose to split the leaf with the largest primal quadric error. In constructing splitting planes, R-Simp will often need to gather normal and midpoint information for every face in a cluster. It also performs a topology check to guarantee disconnected components within a cell are in fact separated by the splitting plane. Our BSP-Tree relies on the eigenvectors of the dual quadric, corresponding to the directions of minimum and maximum point spread, to generate a plane. Using point-set covariance in this manner allows us to construct a plane without requiring access to the geometry and topology of original surface, which would be prohibitively expensive.

While we chose to work with point distribution information for reasons of efficiency, we also believe there are situations which make this approach preferable to constructing splitting planes based on the curvature information contained in the primal quadric. Consider an almost planar surface with high-frequency perturbations running along it. The pattern of normal variation might be unclear because of the bumps, but the point set covariance matrix will still allow a good approximation of the surface in the form of the least-squares best fit plane. At the beginning of the simplification process, encountering an ambiguous pattern of normal variation is almost assured. The first few split planes will be computed with regard to almost all of the faces in the mesh, making it unlikely that a coherent pattern of curvature will be discernible.

The use of quadric quantization to construct a spatial partition allows our algorithm to work on very large meshes, and we emphasize that large, over-tessellated meshes are the intended domain of our method. We do not intend this algorithm to be a replacement for more traditional simplification methods; polygonal models small enough to fit in-core are best handled using other algorithms. Also, as a general caveat, all spatially-based clustering methods tend to produce approximations with altered topology. Generating or closing small holes and creating self intersections are common occurrences; manifold surfaces will likely be approximated by a non-manifold surface. When the approximation is intended for visualization these artifacts tend not to be significant, but they should be considered if the intended application is sensitive to such changes.

5 RESULTS

To evaluate the performance of our algorithm, we implemented both it and the uniform clustering method described by Lindstrom [16]. We should note that our implementation differs from his in that we used the quadric metric as originally formulated by Garland and Heckbert rather then the one described in [18]. Also, we did not implement the singular value decomposition step for robust inversion of quadric matrices that he suggests. Neglecting these details should have little impact on comparative evaluation as

	Vertices		Avg. Error ($\times 10^{-9}$)		Time (sec)	
Model	Input	Output	Adaptive	Uniform	Adaptive	Uniform
Buddha	543,652	5644	68.0	79.0	17	7
Buddha	543,652	41,638	8.1	8.9	24	8
Dragon	437,645	1077	1319.0	1700.0	10	4
Dragon	437,645	5183	70.0	79.0	13	5
Matthew	3,382,866	9280	–	–	102	46
Lucy	14,027,872	14,818	–	–	425	190

Table 2: Comparison of uniform and adaptive clustering. Due to the input size limitations of the error tool, error information is unavailable for St. Matthew and Lucy models.

both of our implementations will be impacted by their lack. Moreover, nothing in our adaptive clustering algorithm would prohibit the implementation of these features.

In order to evaluate the error in an approximation, we created a tool that samples both the original and simplified models at their vertices and computes the distance between the two surfaces at those points. This error metric is similar to the one employed by Hoppe [12] and the tool Metro [5]. The tool begins by computing the distance from each vertex in the original mesh to the simplified mesh. A second pass computes the distance from each vertex in the simplified mesh back to the original mesh. The average of all these distances is then taken to be the average error. Table 2 summarizes the results of our experiments, which were run on a standard Linux PC with an 800 MHz P3 processor, 256 MB of memory, and a SCSI disk.

In general, the adaptive algorithm produced better quality approximations. In the case of the coarse approximations of the dragon and Buddha models, error was reduced by about 20 percent. Finer resolutions mirror this behavior, but error reduction is only around 10 percent in those cases. The time required to simplify adaptively varied from around 2.5 to 3 times as much as that required for uniform clustering. This corresponds to expectations, since the adaptive method effectively simplifies the mesh twice while the uniform method does so only once. The size of the quantization grid used in the adaptive simplification process was larger than the grid used for the uniform clustering by a factor of four in each dimension.

Figure 3 shows how the error was distributed across a Buddha model simplified from 543,652 to 812 vertices by both uniform and adaptive clustering. Increasing error is denoted by a color progression from blue through green and yellow to red. The color scales are the same for both images. It appears that uniform clustering generally introduced more error in regions of high curvature such as the nose and flower. While exhibiting less error in such highly curved areas, adaptive clustering had greater difficulty generating a good approximation of planar surfaces, as witnessed by the error seen across the base of the model.

Both the Lucy and Saint Matthew models are too large to estimate surface error in the manner applied to the Buddha and dragon. A visual comparison of some results are shown in Figure 4 and Figure 5. To highlight the differences, these images are the result of very aggressive simplification, an order of magnitude beyond those shown in [16]. Figure 5 shows two simplified versions of the Lucy, one produced using uniform clustering, the other with our adaptive method. The uniform clustering version displays more artifacts on the wings, both spiking and notching, than the adaptive version does. The spikes are related to a cell enclosing two nearly parallel sheets of the surface, a case in which quadric vertex placement can yield a vertex position outside the cell. The version of uniform clustering proposed by Lindstrom implements a clamping scheme to deal with such situations. Another significant difference between the approximations occurs in the detail around the face of the statue. At this resolution, uniform clustering has removed most of the mouth and nose, and the head has started to sink into the neck. Adaptive clustering is able better able to alter the size and distribution of polygons in the approximation. As a result, the face has retained much more distinct features, particularly at the eyes and neck. In other portions of the statues, the adaptive version produced a better approximation to the torch while uniform clustering displayed better ability handling the sharp border of the base.

In the case of the Saint Matthew approximations, shown in Figure 4, the regularity of the uniform grid can be seen in the approximation. The lines of a checkerboard pattern are visible around the chin and cheeks. The approximation generated by our adaptive method does not exhibit this artifact. Some features, such as the nose, seem sharper in the adaptive version. In both cases, the lack of boundary constraints has caused the mesh to be eaten away along the border.

While perhaps not obvious in these images, the adaptive simplification method has a greater propensity to join spatially separated sections of the mesh than uniform clustering does. In the Lucy approximations for example, the adaptive simplification often joins the raised arm with the wing while uniform clustering rarely does. An even clearer example of this behavior can be seen in the coarsest approximation shown in Figure 1. The cell size of a uniform grid, even without clamping, usually limits how far a representative vertex can drift from the original vertices in a cell. Our adaptive spatial partition offers no such bound, which is in effect the price paid for adaptivity. While variable cell size allows more polygons in areas of high detail and fewer polygons in more planar regions, it also implies that a poor choice of splitting plane can result in a representative vertex moving far from the original mesh. We hope to avoid this artifact in the future by designing more complex rules for the placement of splitting planes.

6 CONCLUSION

We have described an adaptive surface simplification algorithm capable of efficiently producing high quality approximations of massive polygonal models. Our experiments have demonstrated that approximations generated using this algorithm do not display artifacts associated with the imposition of an artificial grid. As a result, the quality of these approximations can exceed the quality of approximations produced by uniform clustering. The execution time remains competitive with that of uniform clustering. The key to the algorithm is the three step process of quadric quantization, partition construction, and simplification. This process yields a simplification hierarchy, making it possible to create a level-of-detail representation or generate a progressive transmission without resorting to a computationally expensive edge contraction algorithm.

We see several opportunities for further research. The quality of our simplified models would be enhanced if we could prevent unnecessary joining of elements in the mesh. We believe this can be accomplished by a more judicious choice of splitting planes, perhaps requiring additional information be accumulated in the quantization phase. Second, the simplification hierarchy implicit in the BSP-Tree produced by the algorithm can be leveraged to provide progressive transmission and perhaps a file format describing a surface based multi-resolution model.

7 Acknowledgements

We would like to thank Marc Levoy and the people working on the Digital Michelangelo Project for making their models available to us.

References

[1] Pankaj K. Agarwal and Subhash Suri. Surface approximation and geometric partitions. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 24–33, 1994. (Also available as Duke U. CS tech report, ftp://ftp.cs.duke.edu/dist/techreport/1994/1994-21.ps.Z).

[2] F. Bernardini, J. Mittleman, H. Rushmeier, and G. Taubin. Case study: Scanning michelangelo's florentine pieta. In *ACM SIGGRAPH 99 Course Notes, Course 4*. ACM SIGGRAPH, August 1999.

[3] Dmitry Brodsky and Benjamin Watson. Model simplification through refinement. In *Proceedings of Graphics Interface 2000*, pages 221–228, May 2000.

[4] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998.

[5] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–74, June 1998. http://vcg.iei.pi.cnr.it/metro.html.

[6] Jihad El-Sana and Yi-Jen Chiang. External memory view-dependent simplification. *Computer Graphics Forum*, pages 139–150, August 2000.

[7] Michael Garland. Multiresolution modeling: Survey & future opportunities. In *State of the Art Report*, pages 111–131. Eurographics, September 1999. http://www.uiuc.edu/~garland/papers.html.

[8] Michael Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University, CS Dept., 1999. Tech. Rept. CMU-CS-99-105. http://www.uiuc.edu/~garland/research/thesis.html.

[9] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Proc.*, pages 209–216, August 1997. http://www.uiuc.edu/~garland/research/quadrics.html.

[10] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical face clustering on polygonal surfaces". In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 49–58, 2001.

[11] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings SIGGRAPH 96*, pages 171–180, 1996.

[12] Hugues Hoppe. Progressive meshes. In *SIGGRAPH '96 Proc.*, pages 99–108, August 1996. http://research.microsoft.com/~hoppe/.

[13] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization 98 Conference Proceedings*, pages 35–42,516, Oct 1998.

[14] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. on Graphics*, 15(3):179–210, 1996.

[15] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The Digital Michelangelo Project: 3D scanning of large statues. *Proceedings of SIGGRAPH 2000*, pages 131–144, July 2000.

[16] Peter Lindstrom. Out-of-core simplification of large polygonal models. *Proceedings of SIGGRAPH 2000*, pages 259–262, July 2000.

[17] Peter Lindstrom and Claudio Silva. A memory insensitive technique for large model simplification. *IEEE Visualization '01*, October 2001.

[18] Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualization 98 Conference Proceedings*, pages 279–286,544, Oct 1998.

[19] Kok-Lim Low and Tiow-Seng Tan. Model simplification using vertex-clustering. In *1997 Symposium on Interactive 3D Graphics*. ACM SIGGRAPH, 1997. http://www.iscs.nus.sg/~tants/.

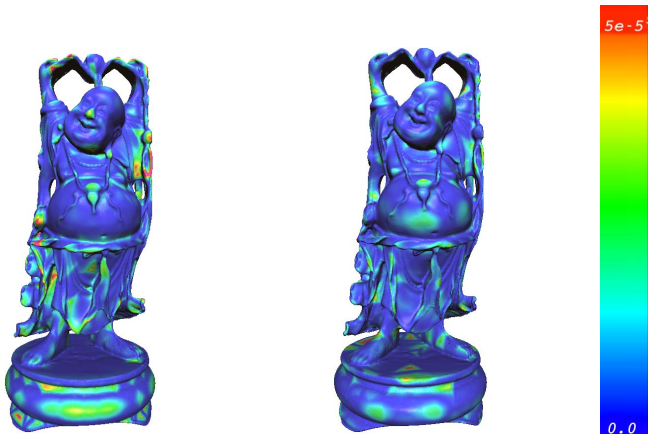
[20] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Proc.*, pages 199–208, August 1997.

[21] Chris Prince. Progressive meshes for large models of arbitrary topology. Master's thesis, University of Washington, 2000.

[22] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, 1993.

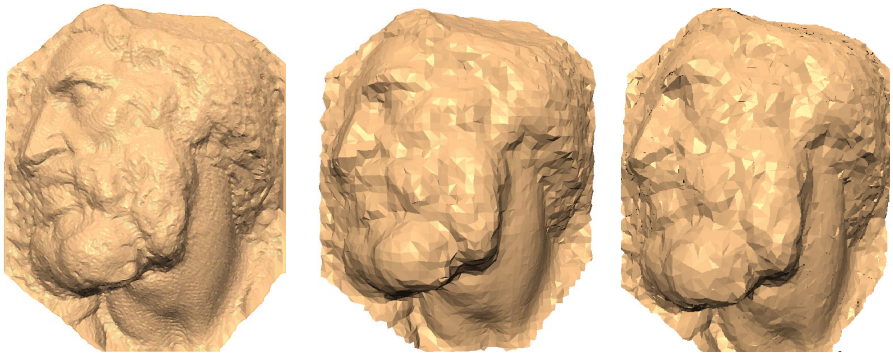
[23] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. *Proceedings of SIGGRAPH 2000*, pages 343–352, July 2000.

[24] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.



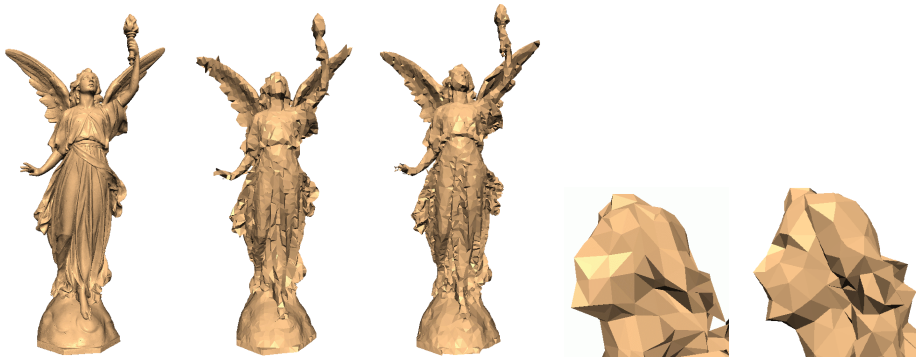
(a) Uniform Clustering (b) Adaptive Clustering

Figure 3: Error distribution on the Buddha model.



(a) Original (b) Uniform Clustering (c) Adaptive Clustering

Figure 4: Face of St. Matthew, simplified from 3,382,866 vertices to 3,225 vertices.



(a) Original (b) Uniform (c) Adaptive (d) Uniform (e) Adaptive

Figure 5: Lucy statue, simplified from 14,027,872 to 5,461 vertices.