Adaptive Vertex Clustering Using Octrees

Scott Schaefer and Joe Warren

Abstract. We present an adaptive vertex clustering approach to out-of-core simplification of polygonal meshes using a dynamic octree. Similar to uniform clustering, our technique utilizes quadratic error functions to position the mesh vertices; however, this new method can resolve vertices to arbitrary resolutions, which allows reproduction of extremely small details and more accurate simplifications. By sorting the vertices of the input mesh, our algorithm dynamically discovers portions of the mesh that are locally complete, which are then available for collapse when the octree grows too large. This adaptive octree is then used to cluster the vertices of the input to generate a simplified mesh. Finally, we show that our method generates the same tree that would be constructed with unbounded memory if our method is given a small amount of space in addition to the size of the output tree.

§1. Introduction

Recent years have seen a substantial increase in extremely large polygonal models. Endeavors such as the Digital Michelangelo project attempt to produce a digital archive of great works of art down to a resolution of a few millimeters. With this resolution, models on the order of tens to hundreds of millions of polygons have been generated. Despite advances in graphics hardware, these models cannot typically be rendered in real-time on a consumer level computer and, therefore, simplification of these models is required. However, traditional simplification algorithms typically process the entire mesh in memory and fail when presented with such a daunting task.

Out-of-core simplification algorithms are a special class of simplification techniques that remove the requirement that the entire model fit into memory. These methods usually utilize disk space instead of memory to store data and only load small portions of the model into memory at

xxx and xxx (eds.), pp. 1–4.

ISBN 0-9728482-x-x

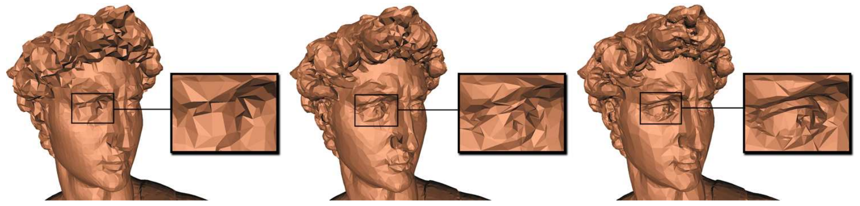

Fig. 1. Close-ups of David's head using three different out-of-core simplification algorithms. Each reduction has the same number of polygons (1.5% of original model) and is flat-shaded. Uniform clustering (left), Shaffer and Garland's adaptive BSP approach (middle), our adaptive octree method (right).

any one time keeping the memory requirements of the algorithm low. Although these techniques do not depend on the size of the input model, many require that the final output model fit into memory. This constraint is not problematic because the goal is to reduce the model to a size that allows more accurate in-core operations to be performed (such as display, further simplification, etc...).

§2. Related Work

There are several different strategies for simplifying massive polygonal models [1, 9, 14, 3, 17]. The method that we will concentrate on in this paper is vertex clustering. Rossignac and Borrel [15] introduced one of the first simplification algorithms that use this technique. Lindstrom [13] later developed an improvement that employed quadratic error functions (QEF's), which were made popular for surface simplification by Garland and Heckbert [7], to position the representative vertex. The algorithm divides the space spanned by a bounding box into a uniform grid. All of the vertices from the original model that are contained within a grid cell are clustered together to form a single representative vertex. This method has become popular for simplifying very large models due to the small memory requirements and extremely fast simplification times.

Unfortunately, uniform clustering cannot reproduce features smaller than a grid cell and may trivially tile large, flat areas with many polygons. To combat these difficulties, Shaffer and Garland [16] introduced an adaptive technique for vertex clustering based on BSP trees. By exploiting an initial uniform clustering phase, their algorithm constructs a BSP tree as the spatial partitioning data structure for clustering vertices. The authors achieve approximately a 20% decrease in average geometric error using an initial uniform clustering grid four times larger in each dimension than Lindstrom's technique. However, the authors do not analyze whether using uniform clustering at the higher resolution grid and then simplifying using an in-core technique would generate the same reductions in error. Also, the authors note that their BSP tree lacks the spatial coherence that uniform clustering provides and, consequently, outliers can connect vertices that are geometrically far apart resulting in poor simplifications.

Garland and Shaffer [8] recently published a multiphase approach to out-of-core surface simplification where uniform clustering is performed on the mesh first and then a mesh is built with the QEF's from uniform simplification attached to the vertices. QSLIM [7], a popular in-core surface simplification tool, is then run on this representation to reduce the number of polygons even further. By combining these two methods, the authors improved running times and generated results comparable to running QSLIM on the original data.

Contributions
The method that we propose is a vertex clustering algorithm related to uniform clustering except we have replaced the uniform grid with an adaptive octree containing a pre-specified number of nodes. This octree is built using in-core memory proportional to the size of the simplified mesh. We also show that our method generates an octree with minimal error resulting in better simplifications than other vertex clustering techniques. Furthermore, the QEF's generated by our method can be used in QSLIM to perform additional simplifications in-core as done in [8].

First we provide an explanation of the vertex sorting phase of our algorithm and the properties that the sorting function must satisfy so that we can safely construct an adaptive octree. Next, we describe our process for constructing the adaptive octree including how we choose the sub-portions of the tree to collapse when the octree grows too large. This construction leads to an analysis of our octree compared to an ideal, in-core vertex clustering technique. Finally, we end with comparisons of our results against other popular vertex clustering techniques.

§3. Vertex Sorting

The first phase in our simplification algorithm is a vertex sorting pass. Without loss of generality, we assume that the mesh is contained in the domain $[0, 1]^3$. We also assume (as does uniform clustering) that the mesh is in a *triangle soup* format where triangles consist of a list of three points in space.

For each polygon $\{v_1, v_2, v_3\}$ in the input format, the method calculates a normal $n = (v_1 - v_2) \times (v_3 - v_2)$ and writes the vertices with the associated normal $\{\{v_1, n\}, \{v_2, n\}, \{v_3, n\}\}$ to disk to be sorted. This sorting process induces a linear ordering of three-dimensional space. We require that the ordering of the vertices is such that if any cube in the octree over the unit domain is split into eight sub-cubes and some enumeration is chosen, then

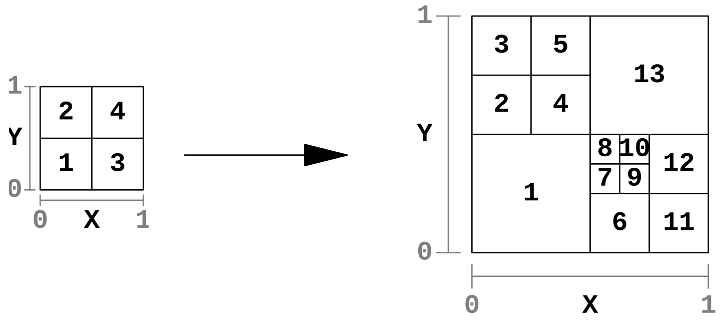

Fig. 2. Enumeration of sub-cubes chosen for sorting (left) and sorted order applied to a quadtree (right).

all of the vertices within sub-cube$_i$ precede all vertices in sub-cube$_{i+1}$. For example, in 2D the key $0.x_1y_1x_2y_2...$ formed by alternating the binary digits of the point $\{0.x_1x_2..., 0.y_1y_2...\}$ satisfies our sorting requirements (and is the ordering used in our implementation). Figure 2 illustrates an example of this sorted order on the nodes of a quadtree.

Any out-of-core sorting algorithm will suffice to sort the vertices; however, a poor choice will affect performance. Our implementation uses a three-dimensional out-of-core merge sort to order the vertices and requires approximately $\frac{8}{7}$ times the disk space needed to store the file with the separated vertices and normals. Lindstrom [12] suggests rsort [11] as an excellent sorting algorithm for these purposes. In terms of disk I/O, this phase of the algorithm is $O(nlog(n))$ so this portion dominates the running time of the simplification algorithm.

§4. Adaptive Vertex Clustering

Once the vertices are sorted, the algorithm constructs an octree to cluster the sorted vertices. We maintain several invariants of the octree. First, the octree may be incomplete in that all eight children are not required to exist. The final property that we maintain is that uncollapsed leaves in the tree store only a single vertex from the original model.(Given the inaccuracy of floating point equality, we require only that the vertices for a node lie within a user specified tolerance from each other.)

Similar to uniform clustering, our technique utilizes quadratic error functions (QEF's) to position the vertices. In our octree, each node contains a QEF to represent the aggregate surface inside of that cell. We define this QEF to be a function of the form of equation 1.

$$E[x] = \sum_i (n_i \cdot (x - p_i))^2 \tag{1}$$

In this equation n_i represents a normal and p_i is a point in space. Uniform clustering never evaluates $E[x]$ and only uses the QEF to position the vertices. However, our adaptive clustering technique will need to calculate the error associated with the QEF. Therefore, we do not use the normal equation format for the QEF, but instead use the QR factorization method presented by Ju et al. [10], which yields a more numerically stable form of $E[x]$.

4.1. Vertex Insertion

Our algorithm takes as a parameter the maximum number of octree nodes available to construct the tree. At the beginning of the octree construction step, the algorithm pushes all available nodes onto a stack from which nodes will be allocated later. Vertices are processed in the order generated by the sorting phase. For each vertex/normal pair, $\{v_i, n_i\}$, the algorithm finds the deepest node in the octree that contains the vertex. If that node is an internal node, then the child containing the vertex does not exist so the child is allocated and the vertex is inserted into that child. Otherwise, the deepest node containing the vertex must be a leaf node.

If the vertex to be inserted is the same as the vertex stored in the leaf, then the algorithm merges the plane equation described by $\{v_i, n_i\}$ into the QEF for that node; otherwise, a split occurs in the tree. The current leaf node becomes an internal node and the vertex previously stored in that node is inserted into the internal node (causing a child to be allocated since none exist) and the QEF is copied into that child. Next, the current vertex is inserted into the new internal node. If the vertex is located in a child that is not present, then the insertion terminates. However, if the vertex is inserted into the same child as the previous vertex, then this process continues until the algorithm separates the vertices into two different nodes. Figure 3 (middle) shows an example of two splits caused by a vertex insertion into a binary tree.

4.2. Node Collapse

During processing, vertex insertion may exhaust the available space in the stack. To free new space for further vertex insertions, the method collapses certain subtrees of the octree to form new leaves. To guide this collapse, we label the current nodes in the octree as either *pending* or *complete*. A pending node is one that lies on the path from the root to the last vertex inserted into the tree. Subtrees whose roots are pending may not be collapsed, since new vertices may later be inserted into these subtrees (leading to poor simplifications). All remaining nodes are complete. Due

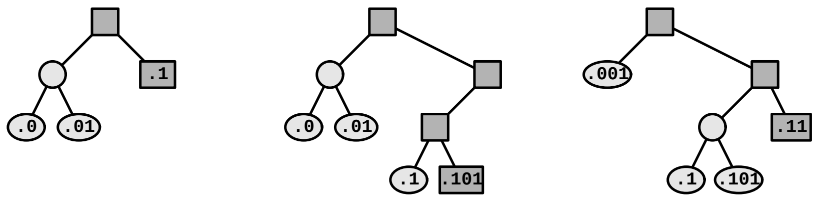

Fig. 3. Construction of a dynamic tree inserting the binary data $\{.0, .01, .1, .101, .11, ...\}$. Squares are *pending* nodes and circles are *complete* nodes. Initial tree after adding first three pieces of data(left), vertex insertion causing two splits (middle), vertex insertion causing collapse (right).

to the sorted vertex order, no vertices will subsequently be inserted into subtrees composed of complete nodes. In figure 3, the left subtree of the root contains all coordinates with key less than .1. Since all subsequently inserted keys are greater than or equal to .1, this subtree is complete and suitable for collapse.

In determining which subtree to collapse, our method chooses the complete subtree whose aggregate QEF has the smallest error. To determine the complete node with minimum error efficiently, the method stores pointers to complete nodes and their error in a heap. All that remains is to determine which nodes are completed during insertion of a particular vertex and insert them into the heap. To that end, we construct a list of internal nodes visited during the insertion of the current vertex (starting with the root node). When inserting this vertex into the octree, we also traverse the list of internal nodes built during insertion of the previous vertex. If at any point the two lists diverge, all nodes in the previous list from that node onward are newly discovered complete nodes and are inserted into the heap.

4.3. Mesh Generation

After the algorithm constructs the dynamic octree, we generate vertices corresponding to the leaves of the octree. The output of our algorithm is a mesh in a topology/geometry format containing a list of vertices and polygons whose elements index into the vertex array. The algorithm generates this vertex array by performing a tree traversal where, at the leaves, vertices are generated at the position that minimizes the error of the QEF and given a unique index in the vertex array. To generate polygons, our method processes the original triangle soup data file again to generate the final polygons for the simplified model. For each vertex in a polygon we find the leaf in the octree containing that vertex and return the index

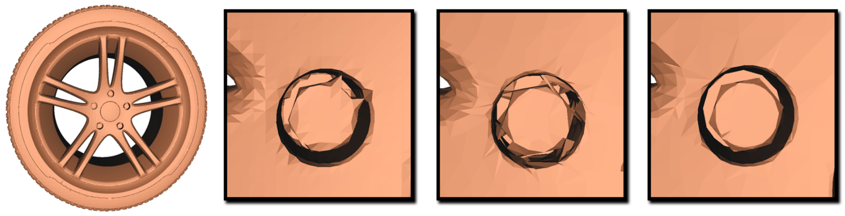

Fig. 4. Simplification of a tire using three different methods to the same number of polygons. Uniform clustering(left), BSP(middle) and our adaptive octree method(right).

stored in that node. If one or more indices of the vertices are equal, the polygon collapsed to either a line or a point during simplification and is discarded; otherwise, the polygon is written to disk.

§5. Analysis

Given an infinite amount of memory, we could construct an octree that has one leaf for each vertex in the original mesh and an associated QEF for that vertex. To generate a simplified mesh with approximately d vertices, we simply collapse nodes with minimal QEF error until the resulting octree T has d nodes. Since the maximal QEF error for the entire octree increases monotonically during collapse, this tree T is optimal in that no other collapsed octree of size d has smaller error. In particular, any octree of size d corresponding to the uniform grid used by Lindstrom's method generates simplified meshes with error greater than or equal to that of T.

We claim that adaptive vertex clustering will reconstruct an optimal tree of size d while generating intermediate octrees of at most size $d+k$ where k is a constant that bounds the maximal number of pending nodes and their immediate children at any point during the simplification. This constant, k, accounts for the maximum number of nodes that cannot be removed due to the existence of pending nodes in the octree. Since pending nodes always lie along a single path from the root of the octree to one of its leaves, the number of pending nodes and their immediate children is bounded by 8 times the depth of octree (at most the number of binary digits in a floating point number). Thus, k is very small in comparison to typical values for d (on the order of 10^2 nodes) and as a result, the space required during adaptive vertex clustering is roughly proportional to the space required to represent the output mesh T.

To prove this claim we note that at any stage in the octree construction phase, nodes are either complete or pending (see figure 3). When a vertex is inserted into the octree there will be at most k nodes pending that cannot be removed. If any of the complete nodes in the tree, of which

Model	Polys	Reduced	Uniform	BSP	Octree
Dragon	871414	74131	3.305/4.0	3.089/10.0	2.586/33.0
Shower	2619676	98167	16.74/13.0	13.82/30.0	10.64/119.0
David*	8254150	126813	98.80/35.0	55.55/75.0	39.97/322.0
Tire*	23980320	808030	29.97/127.0	59.56/201.0	24.65/1172.0

Tab. 1. Simplification results for different models and methods. Results are in the format Average Error $* 10^{-3}$/Time(seconds)

there are at least d, are not part of the tree T, then those nodes can be collapsed to make room for the new vertex without removing any nodes in T (due to the monotonicity of the error function). However, if d complete nodes are all part of T and k nodes are pending, then at some point in the vertex insertion step, the previous list of internal nodes will diverge from the current path. When the paths diverge, all nodes in the tree with depth greater than or equal to the current depth are complete and can be collapsed to reclaim space. Since insertion can only take k nodes, no nodes in T will be removed during the collapse. Therefore, the final tree will contain all of the nodes in the tree T and have maximum error less than or equal to that of T.

§6. Results

To assess the performance of our method, we compared the simplifications produced by our implementations of uniform clustering, Shaffer and Garland's adaptive BSP approach, and our adaptive octree method for several large data sets. Table 1 summarizes the results for several different simplifications of large models. All results are gathered on a Pentium III 3 GHz machine with 2GB of RAM and an ATA/100 IDE hard drive. We use the Metro tool [4] to determine the error of the simplifications. Models with asterisks were too large to analyze with Metro (the program crashes). Instead we extracted the sub-portions depicted in the figures from the original and evaluated the truncated models with Metro. Though the error for the truncated model does not indicate the global simplification error, we include the results to give some indication (other than visual) of how our method performs on these large models. In general we achieved about a 20-50% reduction in error over uniform clustering and approximately a 20% reduction on average over the BSP technique.

In terms of speed, our method is approximately ten times slower than uniform clustering and two to three times slower than Shaffer and Garland's BSP technique mostly due to the sorting phase (see table 2). This slowdown is to be expected though because our algorithm is $O(nlog(n))$ instead of $O(n)$ like the other two methods. We feel this speed is accept-

Model	**Writing Verts**	**Sorting**	**Octree Construction**	**Mesh Generation**
Dragon	2.2s	17.5s	12.8s	0.1s
Shower	6.6s	76.6s	35.7s	0.1s
David	21.7s	213.3s	86.1s	0.2s
Tire	83.9s	780.0s	306.9s	0.8s

Tab. 2. Running times for the different phases of the algorithm on several different models.

able though because out-of-core simplification should be done only once to reduce the model to a manageable size so that in-core simplification algorithms can finish the reduction. Also, the QEF's generated by our reduction algorithm could then be used in QSLIM to perform more accurate in-core reductions as done in [8].

Figures 1, 4 and 5 show three different models (all containing approximately the same number of polygons) reduced using the three algorithms compared here. In each case, uniform clustering does not have a sufficient sampling density to reproduce many of the small features (David's eyes, the tire lug nuts and the center rod of the shower head). Shaffer and Garland's adaptive BSP approach does a better job at reproducing these features, but is still subject to the deficiencies of their original uniform sampling phase. Also, the BSP method generates many small non-manifold vertices produced by poor choices of splitting planes. Since the BSP tree partitions space into arbitrary convex regions, there are no bounds on the length of these convex regions and vertices from distant regions of the mesh can potentially be joined together. In contrast, the cells produced by octree clustering are bounded an can reproduce these fine scale features with a high degree of accuracy.

§7. Conclusions

We have presented a method for adaptive out-of-core simplification using adaptive octrees. By sorting the vertices of the input mesh, our algorithm is able to discover portions of the mesh that are locally complete and construct a dynamic octree using only a constant amount of memory. This adaptive octree is then used to cluster the vertices of the original mesh in order to generate a simplified mesh. We concluded by arguing that if our method were given a small amount of extra space, the octree generated would be the same as if we were allowed infinite memory.

§8. References

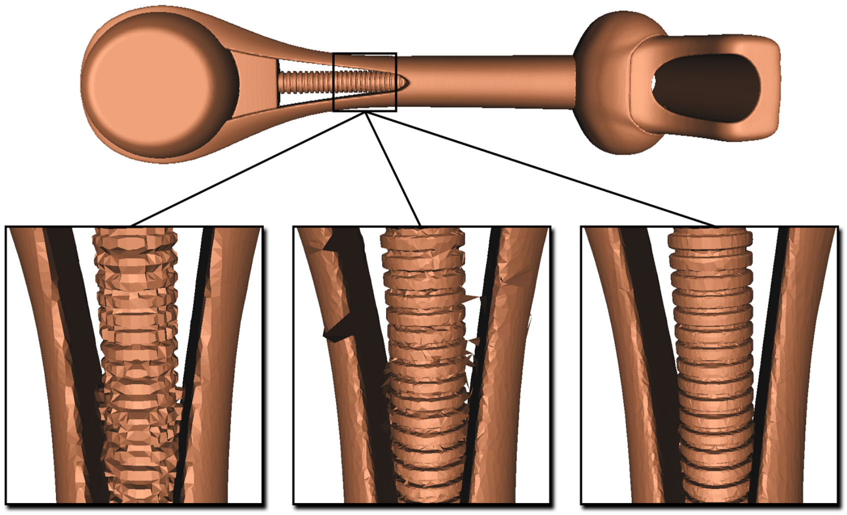

Fig. 5. Simplification of a shower head using three different methods to the same number of polygons. Uniform clustering(left), BSP(middle) and our adaptive octree method(right).

1. Bernardini F., J. Mittleman and H. Rushmeier Case study: scanning michelangelo's florentine pieta. ACM SIGGRAPH 99 Course Notes, Course 8, 1999.
2. Chiang Y., C. Silva and W. Schroeder Interactive out-of-core isosurface extraction. Proceedings of IEEE Visualization, 1998, 167–174.
3. Cignoni P., C. Rocchini, C. Montani and R. Scopigno External memory management and simplification of huge meshes. Transactions on Visualization and Computer Graphics, 2003, 525–537.
4. Cignoni P., C. Rocchini and R. Scopigno Metro: measuring error on simplified surfaces. Computer Graphics Forum **17**, 1998, 167–174.
5. Corrêa W., J. Klosowski and C. Silva iWalk: interactive out-of-core rendering of large models. Technical Report TR-653-02, Princeton University, 2002.
6. Fei G., K. Cai, B. Guo and E. Wu An adaptive sampling scheme for out-of-core simplification. Computer Graphics Forum **21**, 2002, 111–119.
7. Garland M. and P. Heckbert Surface simplification using quadric error metrics. ACM Siggraph Conference, 1997, 209–216.
8. Garland M. and E. Shaffer A multiphase approach to efficient surface simplification. Proceedings of IEEE Visualization, 2002, 117–124.

9. Hoppe H. Smooth view-dependent level-of-detail control and its application to terrain rendering. Proceedings of IEEE Visualization, 1998, 35–42.
10. Ju T., F. Losasso, S. Schaefer and J. Warren Dual contouring of hermite data. ACM Siggraph Conference, 2002, 339–346.
11. Linderman J. rsort and fixcut man pages, 1996.
12. Lindstrom P. and C. Sliva A memory insensitive technique for large model simplification. Proceedings of IEEE Visualization, 2001, 121–126.
13. Lindstrom P. Out-of-core simplification of large polygonal models. ACM Siggraph Conference, 2000, 259–262.
14. Prince C. Progressive meshes for large models of arbitrary topology. Master's Thesis, 2000.
15. Rossignac J. and P. Borrell Multi-resolution 3D approximation for rendering complex scenes. Modeling in Computer Graphics, 1993, 455–465.
16. Shaffer E. and M. Garland Efficient adaptive simplification of massive meshes. Proceedings of IEEE Visualization, 2001, 127–134.
17. Wu J. and L. Kobbelt A stream algorithm for the decimation of massive meshes. Proceedings of Graphics Interface, 2003, 185–192.

Scott Schaefer
Rice University
Houston, TX
`sschaefe@rice.edu`
`http://www.cs.rice.edu/~sschaefe`

Joe Warren
Rice University
Houston, TX
`jwarren@rice.edu`
`http://www.cs.rice.edu/~jwarren`