In Proceedings of ACM SIGGRAPH 2000.

Out-of-Core Simplification of Large Polygonal Models

Peter Lindstrom

Georgia Institute of Technology

Abstract

We present an algorithm for *out-of-core simplification* of large polygonal datasets that are too complex to fit in main memory. The algorithm extends the vertex clustering scheme of Rossignac and Borrel [13] by using error quadric information for the placement of each cluster's representative vertex, which better preserves fine details and results in a low mean geometric error. The use of quadrics instead of the vertex grading approach in [13] has the additional benefits of requiring less disk space and only a single pass over the model rather than two. The resulting linear time algorithm allows simplification of datasets of arbitrary complexity.

In order to handle degenerate quadrics associated with (near) flat regions and regions with zero Gaussian curvature, we present a robust method for solving the corresponding underconstrained least-squares problem. The algorithm is able to detect these degeneracies and handle them gracefully. Key features of the simplification method include a bounded Hausdorff error, low mean geometric error, high simplification speed (up to 100,000 triangles/second reduction), output (but not input) sensitive memory requirements, no disk space overhead, and a running time that is independent of the order in which vertices and triangles occur in the mesh.

1 INTRODUCTION

Polygonal simplification has been a hot topic of research over the last decade, with a vast number of published algorithms. Many of the early simplification algorithms were designed to handle modest size datasets of a few tens of thousands of triangles. As is common in most areas of computing, improvements in processor speed and memory capacity have served merely to promote the production of increasingly larger datasets, and a number of methods, particularly for out-of-core visualization, have been proposed for coping with models that are too large to fit in main memory, e.g. [3, 4, 10]. Following this trend, some of the more recent simplification algorithms have been designed to be memory efficient, and typically handle models with as many as several million triangles. In the last few years, however, there has been an explosion in model size, in part due to improvements in resolution and accuracy of data acquisition devices, such as laser range and CT/MRI scanners. Indeed, submillimeter resolution datasets such as the *Visible Human* [1], which consists of well over 10 billion voxels, and the range scans of Michelangelo's sculptures made independently by research groups at IBM [2] and Stanford University [7] contain up to two billion triangles. These enormous datasets pose great challenges not only for mesh processing tools such as rendering, editing, compression, and surface analysis, but paradoxically also for simplification methods that seek to alleviate these problems. In addition to their large memory consumption, these algorithms also suffer from insufficient simplification speed to be practically useful for simplifying very large meshes. As an example, the "memoryless" simplification scheme proposed by Lindstrom and Turk [8]—one of the fastest and most memory efficient algorithms available—requires a minimum of $160n$ bytes of internal storage to represent an n-vertex model and the necessary edge collapse priority queue. Simplifying a one billion vertex model to a few million triangles using their algorithm would require 160 gigabytes of RAM and, disregarding memory thrashing, would take weeks to complete on a high end workstation!

One might argue that high resolution datasets such as the ones described above are greatly oversampled, and that this problem should be solved more directly during the data acquisition or synthesis stage, e.g. by using adaptive sampling and tessellation during range scanning and isosurface extraction. At best, this simply shifts the problem to an earlier stage of the modeling pipeline, and results not only in a need for specialized tools for each acquisition method, but often raises a number of practical issues. In particular, it places an additional burden on the data acquisitor in terms of deciding how to sample the model and dealing with the difficult issues of registering and integrating different resolution surface patches. In some cases, such a head-on approach is not even practical; one might not know in advance what parts of a surface should be sampled densely, or one might simply wish to retain the model at its full resolution and allow the end-user to resample the model in a manner that suits the given application.

Currently, few algorithms exist for performing high quality out-of-core simplification. One reason for this is that existing in-core methods are difficult to adapt to perform out-of-core simplification, because the majority of them are based on performing simple local operations that rely on having direct access to the connectivity of the mesh. For example, the quality measures associated with the *vertex removal* and *edge collapse* operations typically depend on the triangles surrounding the vertex or edge. Consequently, such methods use large in-core data structures to allow efficient queries of the local connectivity for any given mesh vertex. As mentioned above, such data structures may require hundreds of bytes per vertex, which might even be too large to off-load to disk. Instead, developers of out-of-core simplification algorithms are faced with two alternatives: segmenting the model into multiple pieces and simplifying them individually, or simplifying models using only limited connectivity information, which is the approach taken in this paper.

We propose an efficient and easy to implement surface simplification algorithm that accepts models of arbitrary complexity, and outputs a model that is small enough for in-core mesh processing tools to handle and store internally. The algorithm is based on uniform sampling via vertex clustering, and is enhanced by a novel use of error quadrics, which were originally developed for edge collapse methods (cf. [5, 8]). To our knowledge, our algorithm is one of very few, if not the only one, for doing fast, high quality simplification of arbitrarily large models.

2 PREVIOUS WORK

Polygonal models have grown rapidly in complexity over recent years, yet surprisingly little work has been done on out-of-core simplification. Because most conventional simplification algorithms are not relevant in the context of out-of-core simplification, we will restrict our discussion of related work to methods that are directly related to our algorithm, as well as the few methods that either

e-mail: *lindstro@cc.gatech.edu* web: *http://www.cc.gatech.edu/~lindstro*

perform out-of-core simplification or can be adapted for that purpose. For a comparison of several well-known in-core simplification methods, we refer the reader to [8].

Rossignac and Borrel proposed one of the earliest simplification algorithms [13]. Their algorithm divides the model into cells from a uniform rectilinear grid, and replaces all vertices in a grid cell by a single representative vertex. When clustering vertices together, the majority of triangles degenerate into edges or points and can be discarded, thereby reducing the complexity of the model. Representative vertices are computed by first estimating the impact each vertex has on the visual appearance of the model using a number of ad hoc heuristics. This vertex grading is then used either to compute a weighted average or to select the most important of the original vertices in each cluster as the representative. As alluded to, but not explicitly stated, their algorithm can easily be adapted to work as an out-of-core method. The appeal of this method lies in its simplicity and speed, although the low quality models it produces, due in part to existing vertex positioning schemes, has lead to the use of more sophisticated simplification methods.

Recognizing that Rossignac and Borrel's method is sensitive to translation of the underlying grid, Low and Tan devised a method that uses "floating cells" constructed by sorting the vertices on their importance, and then iteratively letting the most important vertex be the center of a new cluster that absorbs all vertices within an arbitrarily shaped cell volume [9]. While providing higher quality results, one drawback of this approach is that it requires sorting the vertices, which is generally an $O(n \log n)$ procedure, compared to the $O(n)$ running time for Rossignac and Borrel's original scheme.

The *edge collapse* operator has been used extensively in simplification, and is generally considered to produce the highest quality results. Ronfard and Rossignac use edge collapse to coarsen a model while maintaining a list of supporting planes with each vertex [12]. Initially, each vertex is assigned the planes associated with its incident triangles. As two vertices are merged into one by an edge collapse, the new vertex inherits the planes of the merged vertices, and the maximum distance from the new vertex to its supporting planes is used to measure the cost of collapsing the edge. The edges of the model are ordered by increasing cost in a priority queue, and a greedy selection strategy is employed in which the cheapest edge is always collapsed. Inspired by this technique, Garland and Heckbert proposed using quadrics—a succinct encoding of the local surface geometry as a 4×4 symmetric matrix—that allow an efficient computation of the sum of squared distances from a vertex to its supporting planes [5]. Lindstrom and Turk [8] later showed that recomputing the quadrics from scratch in each iteration from the partially simplified surface, and weighting each quadric by the squared triangle area (thus measuring squared displacements in volume) improve the model quality. We use these area-weighted quadrics in our out-of-core simplification algorithm.

Bernardini et al. describe an algorithm that has been specifically designed to perform out-of-core simplification [2]. Their method splits the model up into separate patches that are small enough to be simplified separately in-core using a conventional simplification algorithm. The patch boundaries are left intact to allow the different pieces to be stitched together without cracks after simplification. A new set of patch boundaries is then used as another iteration of simplification is performed, allowing the seams between the previous set of patches to be coarsened. A similar technique was proposed by Hoppe for creating hierarchical levels of detail for height fields [6]. While conceptually simple, the time and space overhead of partitioning the model and later stitching it together adds to an already expensive in-core simplification process, rendering such a method less suitable for simplifying very large meshes.

3 SIMPLIFICATION ALGORITHM

The simplification algorithm presented here is a hybrid of several schemes, including [5, 8, 13]. At a high level, it resembles Rossignac and Borrel's vertex clustering algorithm, but is improved both in execution time and quality by using the quadric error metric introduced by Garland and Heckbert, and later improved by Lindstrom and Turk, for positioning vertices. In particular, our linear time algorithm improves upon [13] by requiring only a single pass over the input model, compared to two or more, and does not use any disk space beyond the input mesh, whereas their algorithm requires an importance value to be stored with each vertex of the input model. In describing our algorithm, we will focus on its novel aspects and assume that the reader is familiar with vertex clustering and quadrics for simplification. We will first describe how the quadrics and representative vertices are computed, and follow with a description of the actual simplification algorithm.

3.1 Quadrics

In order to integrate quadrics with the general vertex clustering scheme, we first make the observation that vertex clustering is a special case of *vertex pair contraction*—a generalization of edge collapse to arbitrary pairs of vertices [5]. That is, merging n vertices within a cluster cell is equivalent to performing any sequence of $n-1$ contractions of pairs of vertices within the cluster until a single vertex remains. As a consequence, we can extend Garland and Heckbert's original scheme from individual vertex contractions to a predefined sequence of such operations. In fact, our algorithm is equivalent to theirs, with the exception that our priority queue is determined by the cluster grid rather than by the local geometry. We use the quadrics from [8], which have proven to give better results in the mean error sense.

Based on [8], we compute for each triangle $t = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ its associated quadric matrix $\mathbf{Q}$ as follows:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{A} & -\mathbf{b} \\ -\mathbf{b}^{\mathrm{T}} & c \end{pmatrix} = \mathbf{n}\mathbf{n}^{\mathrm{T}} \tag{1}$$

$$\mathbf{n} = \begin{pmatrix} \mathbf{x}_1 \times \mathbf{x}_2 + \mathbf{x}_2 \times \mathbf{x}_3 + \mathbf{x}_3 \times \mathbf{x}_1 \\ -[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3] \end{pmatrix} \tag{2}$$

where $\mathbf{n}$ is a 4-vector made up of the area-weighted triangle normal and the scalar triple product of its three vertices. We then distribute $\mathbf{Q}$ to the clusters associated with each of t's three vertices by adding $\mathbf{Q}$ to their quadric matrices. Since $\mathbf{Q}$ is symmetric, and since the scalar c is not used, only 9 scalar values need to be stored with each cluster. After adding up the quadrics of all the triangles in a cluster, we use the block decomposition of $\mathbf{Q}$ above and solve the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ for the "optimal" representative vertex position $\mathbf{x}$. That is, $\mathbf{x}$ is the position that minimizes the sum of squared volumes of the tetrahedra formed by $\mathbf{x}$ and the triangles in the cell.

If a cell contains two nearly parallel surface sheets, the quadrics will sometimes suggest a solution $\mathbf{x}$ that is close to the intersection of the extension of these two surfaces. The solution may in such cases lie far outside the cell itself. We handle these degeneracies by restricting the position of $\mathbf{x}$, either by independently clamping its three coordinates to the cell bounds, or by "pulling" the vertex towards the cell center until it is sufficiently close.

3.1.1 Robust Inversion of Quadric Matrices

In the discussion above, we assumed that the matrix $\mathbf{A}$ is invertible and well-conditioned. In practice, this is often not the case, e.g. if the surface is locally flat or has zero Gaussian curvature. Lindstrom and Turk [8] proposed a partial solution to this problem by ensuring that the problem is overconstrained, and then combining linear constraints that yield a sufficiently large value for the determinant of $\mathbf{A}$. In our case, however, the quadrics yield at most three constraints, and we use a slightly different approach that is able to both diagnose potential problems and also robustly produce the "best" vertex in the sense that $\mathbf{x}$ is chosen such that its distance to the cell center is minimized. That is, $\mathbf{x}$ is the orthogonal projection of the cell center onto the space of all solutions to $\mathbf{A}\mathbf{x} = \mathbf{b}$. We accomplish this by performing a *singular value decomposition* $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$, which for a real symmetric positive semidefinite matrix $\mathbf{A}$ is equivalent to doing an eigenvalue decomposition. This can be done quickly

using a small number of Jacobi rotations [11]. For robustness, we set a lower limit on the singular values and discard (zero) the ones that are negligible:

$$\sigma_i^+ = \begin{cases} 1/\sigma_i & \text{if } \sigma_i/\sigma_1 > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where σ_1 is the largest singular value and ϵ is a threshold parameter currently set to 10^{-3}. The vertex $\mathbf{x}$ closest to the cell center $\hat{\mathbf{x}}$ that satisfies $\mathbf{A}\mathbf{x} = \mathbf{b}$ is then

$$\mathbf{x} = \hat{\mathbf{x}} + \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^{\mathrm{T}}(\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}) \tag{4}$$

which simplifies to $\mathbf{A}^{-1}\mathbf{b}$ whenever $\mathbf{\Sigma}^+ = \mathbf{\Sigma}^{-1}$, i.e. the above equation is used whether $\mathbf{A}$ is ill-conditioned or not, and always yields a numerically robust solution.

3.2 Vertex Clustering

For performance reasons, it is important that the external mesh representation is conducive to the types of mesh queries needed for the given simplification operator. Fortunately, the combination of vertex clustering and quadrics allows commonly used off-line data structures to represent the mesh, such as an *indexed mesh* in which each triangle is a triplet of indices associated with an ordered list of vertex coordinates. By storing the mesh in binary form as fixed-length records, the vertices of a triangle can be fetched from disk indirectly via random access. While such a format is compact, our algorithm requires no connectivity information, and is thus able to operate on a *triangle soup* in which each triangle is represented directly as a triplet of vertex coordinates. The triangle soup representation requires roughly twice as much disk space as the indexed mesh, but typically increases the simplification speed by a factor of 15–20, while also accommodating text file representations. In addition, since our algorithm makes a single pass over the mesh triangles, the triangle soup can be compressed externally and then uncompressed on-the-fly during simplification. The model can even be split up into several files if, for example, it is too large to store on a single disk. We used the triangle soup representation for the results presented in this paper. Similar to Rossignac and Borrel's original clustering algorithm, our algorithm also requires a bounding box for the model, which is divided into a user-specified number of rectilinear grid cells. We anticipate that most data acquisition methods are able to provide such bounds and store the mesh in either of these two formats.

Once the cluster grid has been determined, we proceed by reading the mesh one triangle at a time and incrementally construct an in-core representation of the simplified mesh. It is generally fair to assume that enough memory exists for this simplified mesh since our goal is to produce a mesh coarse enough for in-core tools to process it. Given a triangle $t \in T_{in}$ from the original mesh, we fetch its vertex coordinates. For each vertex v_{in} of t, we construct a hash key from the grid cell that the vertex falls in and do a hash table lookup. This dynamic hash table maps grid cells, or clusters, to the vertices V_{out} in the simplified mesh. If this cell has not been visited, a new vertex identifier v_{out} is created (e.g. using consecutive integers) and the quadric matrix associated with v_{out} is initialized to zero. If two or more of the triangle's vertices belong to the same cluster, then t reduces to an edge or a point, and is discarded. Otherwise, we add it, as a triplet of indices into V_{out}, to the set of simplified triangles T_{out}.

Before proceeding with the next triangle, we compute the quadric matrix $\mathbf{Q}$ associated with t. For each vertex of t, we add $\mathbf{Q}$ to the matrix of the cluster that the vertex belongs to. After the input has been exhausted, we are left with a list of quadrics and a list of triangles. Each quadric corresponds to a cluster of vertices and triangles that share a grid cell, and from the quadric matrix we compute the coordinates for the cluster's representative vertex v_{out} using the procedure described above. The simplification then ends by outputting the simplified mesh (V_{out}, T_{out}) in an appropriate format.

model	$\lvert T_{in}\rvert$	$\lvert T_{out}\rvert$	*RAM (MB)*			*time (h:m:s)*		
			[5]	[8]	*OoCS*	[5]	[8]	*OoCS*
dragon	871,306	244,562	213	134	28	5:31	11:59	0:16
dragon	871,306	113,090	214	134	11	5:55	14:12	0:12
dragon	871,306	47,228	214	134	7	6:06	15:21	0:10
buddha	1,087,716	204,750	250	166	26	7:13	16:58	0:17
buddha	1,087,716	62,354	251	166	8	7:35	19:19	0:12
blade	28,246,208	507,104	-	3,185	63	-	12:37:25	5:02
statue	386,488,573	3,122,226	-	-	366	-	-	1:59:20

Table 1: Simplification results of running QSlim [5], Memoryless Simplification [8], and the out-of-core method (OoCS). All results were gathered on a 195 MHz R10000 SGI Origin with 4 GB of RAM and a standard SCSI disk drive.

4 RESULTS AND DISCUSSION

To evaluate the performance of our algorithm, we include results of simplifying four large polygonal datasets: a buddha, a dragon, and a model of Michelangelo's St. Matthew statue created by researchers at Stanford using a range scanner, as well as a turbine blade model which was extracted from volume data as an isosurface. We applied two levels of Loop subdivision to the blade model to increase its triangle count by a factor of 16, thus making it more challenging to simplify. Table 1 includes the triangles counts, memory usage, and timing results of simplifying these models using our method as well as the in-core methods presented in [5, 8]. While being much more memory efficient than these two methods, our new algorithm is also orders of magnitude faster. Note that the reported memory usage is consistently higher than our implementation's theoretical usage of 63 to 72 $|T_{out}|$ bytes,[1] as the former includes freed memory not reclaimed by the operating system.

Figures 1a–c show the original buddha model and two out-of-core simplified models. Notice how the models in 1a and 1b are virtually indistinguishable, while some blocking artifacts appear in 1c, yet most details are still present. Figures 2b–d show several simplifications of the dragon model. We here compare our vertex positioning scheme based on quadrics against 1) using the mean of a cluster's vertices and 2) the vertex grading scheme of Rossignac and Borrel that chooses the most important vertex, and which has been improved using the technique in [9]. Fine details near the jaws, neck, and hind leg are washed out by the vertex averaging scheme, and the ridge along the back has lost its sharpness. The model produced by vertex grading has a more choppy appearance with loss of detail in the face. Finally, Figures 3a and 3b show close-ups of the face of the St. Matthew statue covering less than 15% of its overall height. This complex model consists of nearly 400 million triangles, and could only be simplified using our out-of-core method. Even after a reduction by a factor of 100, many fine details such as the chisel marks are still preserved.

While the quality of our method is high in comparison with other vertex clustering schemes, it does not perform adaptive sampling of the model, and often produces models that can be further coarsened in areas of low curvature with little loss in quality. For applications that require extreme reduction and very high visual quality, our algorithm can be used as a fast preprocessing step that produces a model with a few hundred thousand triangles, which can then be further simplified by a slower in-core simplification algorithm.

We envision several avenues for future research. As suggested in [13], adaptive sampling can be handled using hierarchical simplification, in which cells are recursively merged in less detailed regions. The idea is to allow quadrics to be merged wherever they "agree" on the local surface characterization. We also believe that the quadric information can be used to improve the connectivity of the mesh, for example by swapping edges in a manner that would reduce the associated quadratic function. Finally, it would be possible to directly integrate our algorithm with the popular *marching cubes* algorithm for isosurface extraction, thereby combining isosurfacing and simplification into a single step, and eliminating the need to output an overly complex intermediate isosurface.

[1] The theoretical memory usage varies with the size and load of the dynamic hash table.

Acknowledgements

I would like to thank Marc Levoy and the people working on the Digital Michelangelo Project for providing the St. Matthew dataset, and Greg Turk, Jarek Rossignac, F. S. Nooruddin, and Gabriel Taubin for valuable comments and suggestions.

1a. Original buddha. 1,087,716 triangles. 1b. OoCS 204,750 triangles. 1c. OoCS 62,354 triangles.

References

[1] ACKERMAN, M. J. The Visible Human Project. In *Proceedings of the IEEE*, 86(3), March 1998, pp. 504–511. Project URL: *http://www.nlm.nih.gov/research/visible*.

[2] BERNARDINI, F., MITTLEMAN, J., and RUSHMEIER, H. Case Study: Scanning Michelangelo's Florentine Pietà. In *ACM SIGGRAPH 99* Course Notes, Course 8, August 1999. Project URL: *http://www.research.ibm.com/pieta*.

[3] BERNARDINI, F., MITTLEMAN, J., RUSHMEIER, H., SILVA, C., and TAUBIN, G. The Ball-Pivoting Algorithm for Surface Reconstruction. In *IEEE Transactions on Visualization and Computer Graphics*, 5(4), October–December 1999, pp. 349–359.

[4] CHIANG, Y.-J., SILVA, C. T., and SCHROEDER, W. J. Interactive Out-of-Core Isosurface Extraction. In *IEEE Visualization '98* Proceedings, October 1998, pp. 167–174.

[5] GARLAND, M. and HECKBERT, P. S. Surface Simplification using Quadric Error Metrics. Proceedings of SIGGRAPH 97. In *Computer Graphics* Proceedings, Annual Conference Series, 1997, ACM SIGGRAPH, pp. 209–216.

[6] HOPPE, H. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. In *IEEE Visualization '98* Proceedings, October 1998, pp. 35–42.

[7] LEVOY, M. The Digital Michelangelo Project. In proceedings of the *Second International Conference on 3D Digital Imaging and Modeling*, October 1999, pp. 2–11. Project URL: *http://graphics.stanford.edu/projects/mich*.

[8] LINDSTROM, P. and TURK, G. Fast and Memory Efficient Polygonal Simplification. In *IEEE Visualization '98* Proceedings, October 1998, pp. 279–286.

[9] LOW, K.-L. and TAN, T.-S. Model Simplification using Vertex-Clustering. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, April 1997, pp. 75–82.

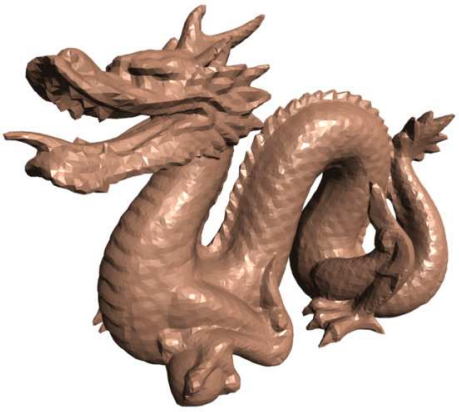
[10] PHARR, M., KOLB, C., GERSHBEIN, R., and HANRAHAN, P. Rendering Complex Scenes with Memory-Coherent Ray Tracing. Proceedings of SIGGRAPH 97. In *Computer Graphics* Proceedings, Annual Conference Series, 1997, ACM SIGGRAPH, pp. 101–108.

[11] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., and FLANNERY, B. P. Numerical Recipes in C: The Art of Scientific Computing, Second Edition. Cambridge University Press, 1992, pp. 408–412.

[12] RONFARD, R. and ROSSIGNAC, J. Full-Range Approximation of Triangulated Polyhedra. Proceedings of Eurographics 96. In *Computer Graphics Forum*, 15(3), August 1996, pp. 67–76.

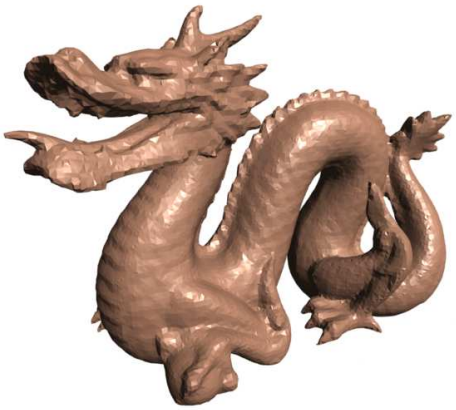
[13] ROSSIGNAC, J. and BORREL, P. Multi-Resolution 3D Approximations for Rendering Complex Scenes. In *Modeling in Computer Graphics*, edited by B. Falcidieno and T. L. Kunii, Springer-Verlag, 1993, pp. 455–465.

2a. Original dragon. 871,306 triangles.

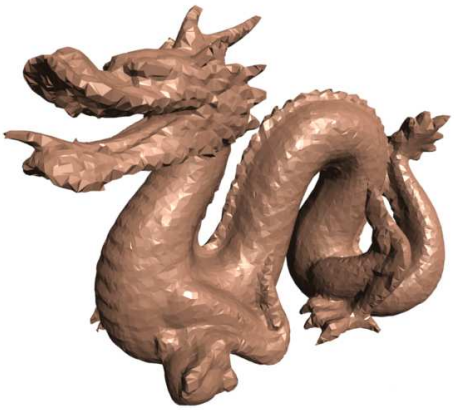


2b. OoCS/Quadrics. 47,228 triangles.

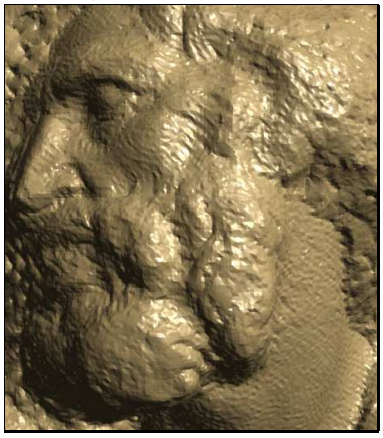
3a. Original statue. 386,488,573 triangles.



2c. OoCS/Vertex mean. 47,228 triangles.



2d. OoCS/Vertex grading. 47,228 triangles.



3b. OoCS. 3,122,226 triangles.