SCCG IEEE proceedings, ISBN 0-7695-1215-1, pp.22-29, 2001

Parallel Triangular Mesh Decimation Without Sorting

Martin Franc[1], Václav Skala[2]
Department of Computer Science and Engineering
University of West Bohemia in Plzen

Abstract

The common task in computer graphics is to visualise models of real world objects. These models are often represented by triangular mesh, which can be very large and complex (thousands or million triangles). Since we want a fast and interactive manipulation with the models, we need either to improve our graphics hardware or to find any method for reducing number of triangles in the mesh. We present here a fast algorithm for triangular mesh reduction based on the principle of mesh decimation.

We present an efficient and stable algorithm for triangular mesh simplification in parallel environment. We use a method based on vertex decimation and our original super independent set of vertices to avoid critical sections.

1 Introduction

Due to wide technological advancement on field of computer graphics during last years, there is real expansion of applications dealing with models of real world objects. For the representation of such models a triangular mesh is commonly used. With growing demands on quality and complexity of computations we can meet models of which surfaces contain hundreds thousand or millions triangles. The models are usually produced by:

- 3D scanners, computer vision and medical visualisation systems, which can produce models of real world objects.
- CAD systems, usually producing complex and high detailed models.
- Surface reconstruction methods or methods for iso-surface extraction, which give us models with very dense polygonal mesh, usually with regular vertices arrangement.

Since the visualisation of large and complex models has high demands of computers performance, the techniques for mesh simplification have been developed. The aim of such techniques is to create an approximation of the original model preserving important details of the object shape. However these techniques are often very slow and therefore unsuitable for large datasets.

Our goal was not to make an algorithm, which will produce high quality surface in sense of approximation error. We wanted to develop a fast and easy to implement algorithm for simplification of large and complex triangular meshes. Our method is built on known decimation techniques and increased the efficiency by parallel implementation. Proposed algorithm has been tested on large datasets.

This paper is structured as follows: In section 2 we discuss a previous work, basic techniques for the mesh simplification especially decimation and a general overview of our previous approach. We present our improvements to the method as well as data structures used and our new algorithm in section 3. Section 5 introduces our results. Time comparison, approximation quality and some examples of reduced models.

{marty|skala} @kiv.zcu.cz
http://home.zcu.cz/~{marty|skala}
This work was supported by
[1]The Ministry of Education of The Czech Republic
– project MSM 235200002
[2]Academy of Sciences of The Czech Republic
– project A 2030801

2 Previous work

2.1 Mesh decimation

Mesh decimation methods are simplification algorithms that start with a polygonisation (typically a triangulation) and successively simplify it until the desired level of approximation is achieved. Most of decimation algorithms fall into one of the three categories, discussed below, according to their decimation technique.

One of the most used methods is vertex decimation, an iterative simplification algorithm originally proposed by Schroeder [1]. In each step of decimation process, all vertices are evaluated according to their importance. The least important vertex is selected for removal and all the facets adjacent to that vertex are removed from the model and the resulting hole is triangulated.

Since the triangulation requires a projection of the local surface onto a plane, these algorithms are generally limited to manifold surfaces. Vertex decimation methods preserve the mesh topology as well as a subset of original vertices.

Other subset of decimation techniques is pointed to the elimination of the whole edge. The principle is similar to the vertex decimation. When the least important edge if found, its length is contracted to zero and triangles, which degenerate to an edge, are removed. Hoppe [2] was the first who has used edge contraction as the fundamental mechanism accomplishing surface simplification. It is also necessary to evaluate the importance of edges before the contraction. One of the best-known techniques [3] uses *quadric error metrics* for the edge (vertex pairs) evaluation.

Unless the topology is explicitly preserved, edge contraction algorithms may implicitly alter the topology by closing holes in the surface.

Techniques, which eliminate either one triangle or any larger area, belong to the last group. These methods delete several adjacent triangles and triangulate their boundary. In case of one triangle, this one is deleted together with three edges and the neighbourhood is triangulated. The evaluation of the reduced elements requires more complex algorithms, in these methods.

2.2 Our approach

Each of the above mentioned approaches have their advantages and disadvantages [4]. We have tried to extract the advantages of all approaches as will be presented in the following part.

We have started with vertex decimation methods and used the Schroeder's approach [1] because of its simplicity and generality in meaning of vertex importance evaluation, and combine it with edge contraction. The methodology of vertex decimation is in fact closely related to the edge contraction approach. Instead of the vertex elimination and arising hole triangulation, one of adjacent edges is contracted, thus we obtain a new triangulation automatically. For the contraction we use the edge that goes out from the eliminated vertex and causes the minimal area of a new surface.

A basic idea that has been used in [6] recently is that decimation by deleting an independent set of vertices (no two of which are joined by an edge, see) can be run efficiently in parallel. The vertex removals are independent and they leave one hole per one deleted vertex, which can be triangulated independently. This decreases the program complexity and run time significantly. We used a technique [6] when we assign an importance value to each vertex, then select an independent set to delete by choosing vertices of the lowest importance.

According to our special data structures used, we have to apply more strict rules on independent set of vertices[3]. Therefore we defined *super* independent set of vertices [7]. Two vertices are in the *super* independent set if all triangles that share them do not share any other vertex from that set, Figure 1.

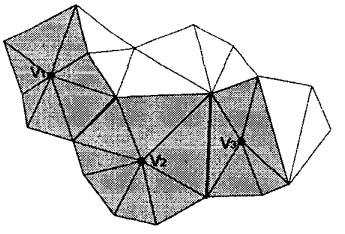

Figure 1: Vertices v1, v2, v3 can belong to the independent set of vertices; vertices v1 and v3 belong to the super independent set of vertices.

To construct a *super* independent set from an assignment of importance values we go through all the vertices in order of their importance and take a vertex if it fit to previous condition.

If we summarize the ideas from paragraphs above we get a fundamental algorithm [7].

[3] Otherwise there will be critical sections in a parallel code and the overhead rapidly decrease the algorithm performance [7].

1. Evaluation of importance of all the vertices (parallel).
2. Sort them according to their importance (sequential/parallel).
3. Create the super independent set of vertices (sequential).
4. Decimate (remove) vertices in the super independent set (parallel).
5. Repeat steps 1 to 4 until desired reduction achieved.

It is obvious that the critical points of the algorithm are the sequential parts (steps 2 and 3). Since we cannot efficiently parallelize the creation of the super independent set of vertices, we substitute an ineffective vertices sorting.

3 Improvements

3.1 Main idea

It is necessary to point to the fact that independent set inhibits vertex importance! Because if we have several least important neighbouring vertices, only one of them will be chosen to the independent set of vertices in one iteration. The rest will be ignored and instead of them other more important vertices will be removed from the mesh.

Considering this we found that it is not necessary to have vertices exactly sorted, so we can use another mechanism, e.g. bucketing function.

The second point came out from the look to the histogram of vertices importance for several datasets, see Figure 2.

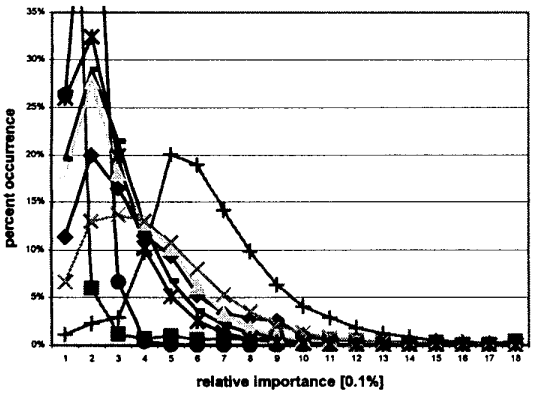

Figure 2: Vertices importance histogram.

It is obvious that 80-90% of vertices have the importance below 1% of the maximum importance value. Therefore we decided to create a super independent set of vertices from vertices with their importance under some threshold only. This approach seemed to introduce big errors in resulting approximation at the first view. But if we consider the behaviour of the algorithm that uses the super independent set of vertices, we find that the approximation quality is sufficient as the experiments proved.

3.2 Data structure used

To improve the efficiency of our parallel algorithm we also used a special data structure. To store the information about vertices we use a data structure similar to a winged edge. With each vertex is kept the number (vertex degree) and indexes of triangles that share the vertex, see Figure 3.

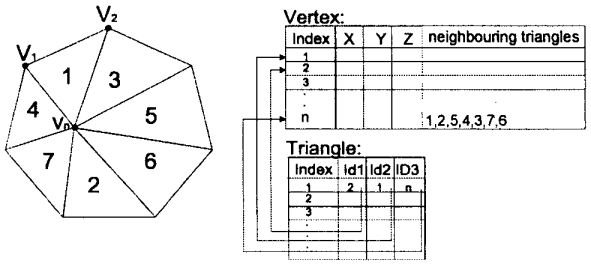

Figure 3: An illustrating scheme of our data structures.

Due to these data structures used we can create the independent set in $O(d*n)$ time, where n is the number of vertices and d is the vertex degree ($d=6$ on an average).

3.3 Resulting algorithm

Having described the ideas upon which our method is based, we can present now the resulting algorithm:

1. Vertices evaluation – vertex type classification and importance computation (parallel).
2. Vertices thresholding and super independent set of vertices creation (sequential).
3. Vertices removal (decimation) – edge importance evaluation, triangulation (parallel).
4. Repetition of steps 1-3 until desired amount of triangles reached.

Parallel parts run as threads, where each thread has its own part of vertices to process. More details can be found in [7].

4 Experimental results

In this section we present results of our experiments that compare a speed-up and approximation error obtained with different data sets, using different number of processors.

We have used several large data sets but we mention experimental results only with 7 different data sets, see Table 1.

Model name	No. of triangles	No. of vertices
Teeth	58,328	29,166
Bunny	69,451	35,947
Horse	96,966	48,485
Bone	137,072	60,537
Hand	654,666	327,323
Dragon	871,414	437,645
Happy Buddha	1,087,716	543,652
Turbine blade	1,765,388	882,954

Table 1: Data sets used.

We made our experiments on DELL Power Edge 8450 – 8xPentium III, cache 2MB, 550MHz, 2GB RAM, running on the Windows 2000.

4.1 Speed-up comparison

Figure 4 shows a graph of the speedup comparison. The speedup a is computed from total times (sequential and parallel parts of the algorithm together) using expression (1).

$$a = \frac{time_1}{time_N}, \qquad (1)$$

where N=1..8 is a number of processors used and $time_N$ is the time obtained if N processors are used.

4.2 Time comparison

The comparison of vertices sorting and vertices thresholding approaches can be seen on Figure 5. On Figure 6 is shown the actual runtime (in percent) of the various algorithm steps for *Happy Buddha* model. You can compare the ratio for old and new method. Notice that the creation of *super* independent set seems to be limiting and the maximal speedup of this approach is approximately 15.

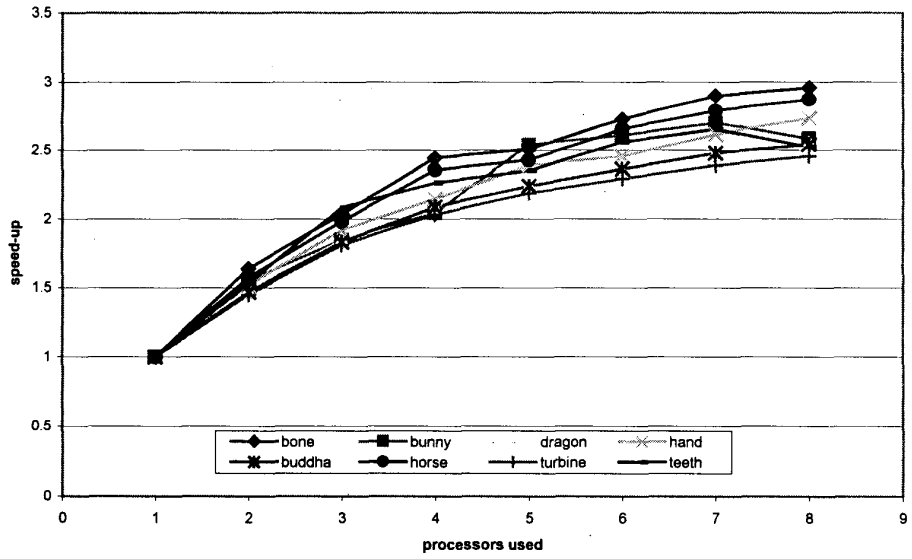

Figure 4: The speed-up of total computation (total time), parallel and sequential parts together; the acceleration is computed for several models of different amount of triangles.

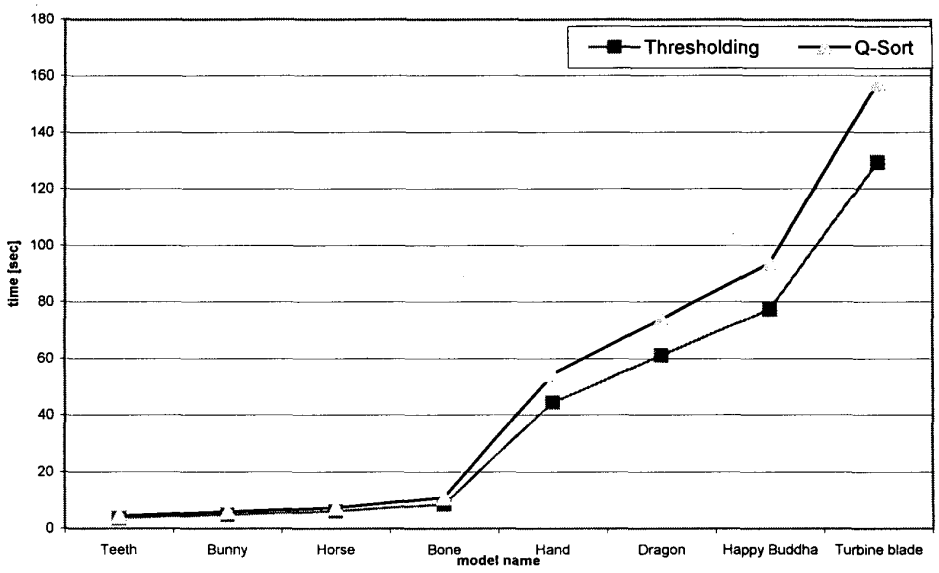

Figure 5: Time comparison of old method with vertices sorting (Quick-Sort) and new approach with vertices thresholding.

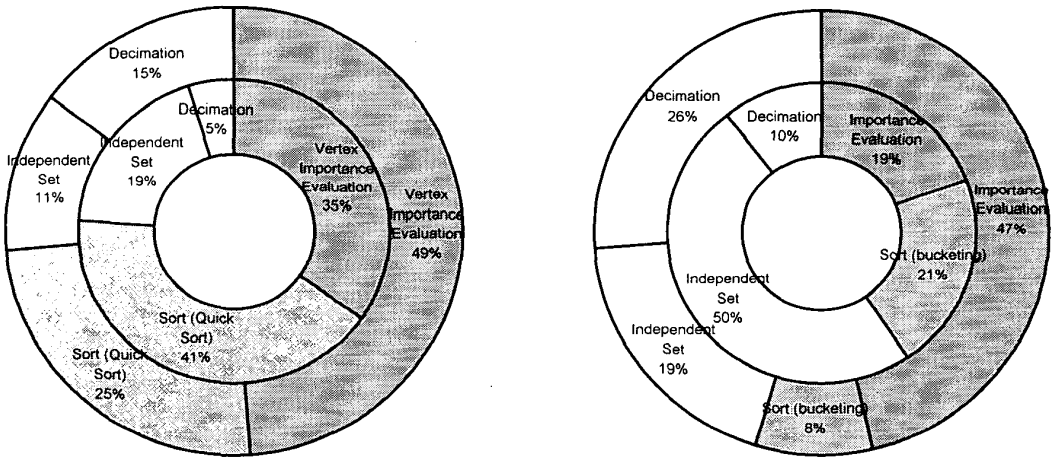

Figure 6: The time ratio of various parts of the algorithm for Happy Buddha model with one (outside) and eight (inside) processors used (vertices sorting on the left, vertices thresholding is on the right side).

Unfortunately we cannot compare achieved times directly with others due to the different platforms used. To make the results roughly comparable at least, we use the official benchmarks presented by SPEC as shows Table 2, where η presents the superiority of DELL computer against the SGI. Table 3 presents our results according to results obtained recently taking the ratio η into the consideration.

Benchmark test / machine	SGI R10000	DELL 410 Precision	η (DELL/SGI)
SPECfp95	8.77	13.1	1.49
SPECint95	10.1	17.6	1.74

Table 2: Benchmark test presented by Standard Performance Evaluation Corporation.

Algorithm	Decim. time from 69.451 to 1.000 triangles [sec]
Presented algorithm	3.7 * k = 3.7* 1.49 = **5.51**
Garland [3]	10.4
Lindstrom & Turk [8]	2585
Hoppe [9]	500
JADE [10]	325

Table 3: Rough comparison.

4.3 Amdahl's law

The experiments proved that the method is stable according to the number of processors used and all the results meet the Amdahl's law (2) perfectly.

$$a = \frac{1}{(1-p)+\frac{p}{N}}, \qquad (2)$$

and therefore

$$p = \frac{N*(1-a)}{a*(N-1)}, \qquad (3)$$

where a is the speed-up, p is potentially parallel code and N is the number of processors used.

The value of potentially parallel code is independent from the number of processors used, see Table 4 and for the large model *Happy Buddha* the value p = 0.69 was reached for the whole algorithm.

	Number of processors (threads) used							
	1	2	3	4	5	6	7	8
$a_{(s)}$	1	1.29	1.43	1.52	1.57	1.61	1.64	1.66
$a_{(t)}$	**1**	**1.46**	**1.83**	**2.08**	**2.23**	**2.36**	**2.47**	**2.53**
$p_{(s)}$	X	0.45	0.45	0.45	0.45	0.45	0.46	0.46
$p_{(t)}$	**X**	**0.63**	**0.68**	**0.69**	**0.68**	**0.69**	**0.69**	**0.69**

Table 4: The comparison of speed-up and potentially parallel code between method using vertices sorting (s) and method with vertices thresholding (t); computed for the Happy Buddha model.

Figure 7 shows the ratio of parallel code (3) of the decimation part according to the number of triangles, for different number of processors used.

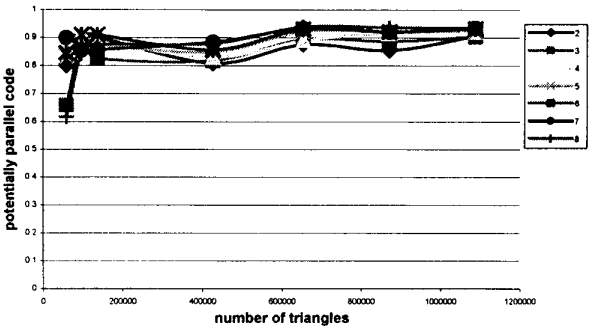

Figure 7: The ratio of the parallel code for decimation part.

4.4 Approximation error

The quality of an approximation can be measured by several approaches. One of the possible ways is to compute a geometric error using E_{avg} metric [11] (4,5) derived from Hausdorff distance:

$$E_{avg}(M_1, M_2) = \frac{1}{k_1 + k_2}\left(\sum_{v \in X_1} d_v^2(M_2) + \sum_{v \in X_2} d_v^2(M_1)\right), \qquad (4)$$

$$d_v(M) = \min_{w \in I'(M)} \|v - w\| \qquad (5)$$

where M_1 and M_2 are original and reduced model, k_1 and k_2 are numbers of vertices on each model, X_1 and X_2 are subsets of vertices in M_1, M_2.

Since our method keeps the subset of original vertices, we can use more simple formula (6):

$$E_{avg}(M_1, M_2) = \frac{1}{k_1}\sum_{v \in X_1} d_v^2(M_2), \quad X_1 \subset P(M_1) \qquad (6)$$

where k_1 is original number of vertices, $P(M_1)$ is a set of original vertices and $d_v(M_2)$ is the distance between original and reduced set of vertices.

Figure 8 presents a comparison of error measurement of the proposed algorithm and M. Garland's method [11] on a Bunny model. However, such a comparison is not very accurate because Garland uses edge collapse procedure and changes vertex positions.

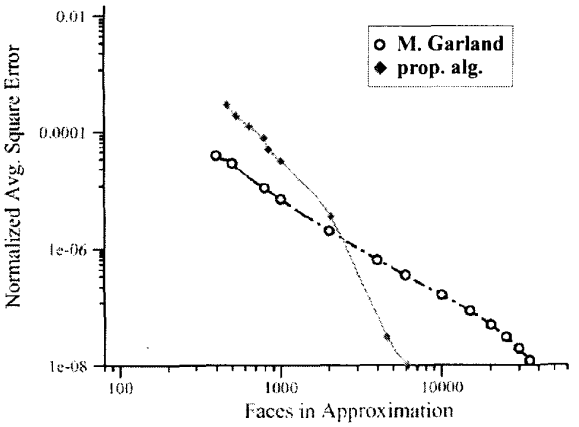

Figure 8: Approximation error comparison.

Examples of reduced models show figures 9 and 10.

5 Conclusion

We have described a new algorithm for parallel triangular mesh decimation without vertices sorting.

The algorithm combines vertex decimation method with the edge contraction to simplify object models in a short time. To improve the efficiency of our parallel algorithm we have used the *super independent* set of vertices and substitute vertices sorting algorithm by fast vertices thresholding. The disadvantage of present implementation is the sequential creation of the *super* independent set of vertices, which is limiting for maximum speed-up about 15. The proposed method proved its stability according to the number of processors and the size of the data set used.

In future we expect to remove independent set creation step using a hash function and also to improve the hole re-triangulation method, which could be controlled by the local error measurement.

Acknowledgements

The authors would like to thank all who contributed to this work, especially to colleagues, MSc. and PhD. Students at the university of West Bohemia in Plzen who have stimulated this work. This paper benefits from several discussions with them a lot. We would also like to thank to DELL Computer Czech Republic for enabling us to carry out all the experiments on their 8-processor computer type.

We also benefited from Cyberware.com model gallery and from the large model repository located at Georgia Institute of Technology;
URL: http://www.cc.gatech.edu/projects/large_models.

References

1. W. Schroeder, J. Zarge, W. Lorensen. *Decimation of Triangle Meshes.* In SIGGRAPH 92 Conference Proceedings, pages 65-70, July 1992.

2. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. *Mesh optimization.* In SIGGRAPH 93 Conference Proceedings, pages 19-26, 1993.

3. M. Garland, P. Heckbert. *Surface Simplification Using Quadric Error Metrics.* In SIGGRAPH 97 Conference Proceedings, 1997.

4. M. Garland. *Multiresolution Modeling: Survey & Future Opportunities.* In the SIGGRAPH 97 course notes, 1997.

5. D. Kirkpatrick. *Optimal Search in Planar Subdivisions.* SIAM J. Comp., pages 12:28-35, 1983.

6. B. Junger, J. Snoeyink. *Selecting Independent Vertices for Terrain Simplification.* In WSCG 98 Proceedings, Plzen University of West Bohemia, pages 157-164, February 1998.

7. M. Franc, V. Skala. *Parallel Triangular Mesh Decimation.* In SCCG 2000 Conference Proceedings, Comenius University Bratislava, May 2000.

8. P. Lindstrom, G. Turk. *Fast and memory efficient polygonal simplification.* IEEE Visualization 98 Conference Proceedings, 1998.

9. H. Hoppe. *Progresive meshes.* SIGGRAPH '96 Proceedings, 1996.

10. A. Ciampaliny, P. Cigony, C. Montani, R. Scopigno. *Mutiresolution decimation based on global error.* The Visual Computer, 1997.

11. M. Garland. *Quadric-Based Polygonal Surface Simplification.* PhD Thesis, School of Computer Science, Carnegie Mellon University, 1999.

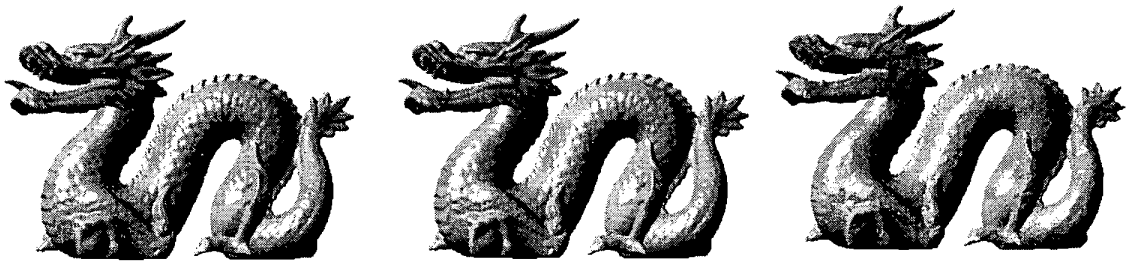

Figure 10: A dragon model (courtesy GaTech) at different resolutions; the original model with 871,414 triangles on the left, reduced to approx. 430,000 triangles in the middle and 87,000 triangles approximation on the right.

Figure 9: A teeth model (courtesy Cyberware); an original model with 58.328 triangles on the left, reduced with vertices sorting method to 2.736 triangles in the middle and reduced by proposed method to 2.730 triangles on the right side.