Fast Algorithm for Triangular Mesh Simplification Based on Vertex Decimation

Martin Franc, Vaclav Skala

Department of Computer Science and Engineering
University of West Bohemia
Univerzitni 8, 306 14 Pilsen, Czech Republic
{marty|skala}@kiv.zcu.cz http://herakles.zcu.cz

Abstract. A common task in computer graphics is the visualisation of models of real world objects. These models are very large and complex and their surfaces are usually represented by triangular meshes. The surface of complex model can contain thousands or even million of triangles. Because we want fast and interactive manipulation with these models, we need either to improve our graphics hardware or to find a method how to reduce the number of triangles in the mesh, e.g. mesh simplification. In this paper we will present a fast algorithm for triangular mesh reduction based on the principle of vertex decimation.

Introduction

Due to the wide technological advancement in the field of computer graphics during the last few years, there has been an expansion of applications dealing with models of real world objects. For the representation of such models polygonal (triangular) meshes are commonly used. With growing demands on quality, the complexity of the computations we have to handle models having hundreds thousands or perhaps even millions of triangles. The source of such models are usually 3D scanners, computer vision and medical visualisation systems, which can produce models of real world objects. CAD systems commonly produce complex and high detailed models. Also there are surface reconstruction or iso-surface extraction methods, that produce models with a very high density of polygonal meshes displaying almost regular arrangement of vertices.

In all areas which employ complex models there is a trade off between the accuracy with which the surface is modelled and the time needed to process it. In attempt to reduce time requirements, we often substitute the original model with an approximation. Therefore, techniques for simplification of large and highly detailed polygonal meshes have been developed. The aim of such techniques is to reduce the complexity of the model whilst preserving its important details.

We shall present a new fast and simple algorithm for the simplification of very large and complex triangular meshes (hundreds of thousands of triangles). The algorithm is based on the combination of commonly used decimation techniques.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2330, pp. 42–51, 2002.

This paper is structured as follows: In section 2 we discuss our previous work and fundamental techniques for the mesh simplification especially decimation. We shall also mention the aspect of parallelization. This section includes an overview of the simplification techniques we recently used. The bucketing (hash) function is described in section 3 and it is used to avoid vertex sorting in the decimation process in order to decrease algorithm complexity. In section 4 we present our new approach in detail – main hypothesis, data structures and the algorithm. Section 5 presents results obtained and section 6 provides a conclusion.

Our Previous Work

As already mentioned, our approach is based on the vertex decimation. We chose the vertex decimation because of its simplicity, stability and fast processing. This method is also easy to parallelize.

Decimation

In general, decimation techniques can be divided into three main categories according to the mesh primitives, which the method is dealing with. Therefore we recognize the following cases:

- Vertex decimation,
- Edge decimation (contraction or collapse),
- Triangle (patch) decimation.

The principle of all the aforementioned methods is similar. Initially we have to evaluate the importance of the primitive (vertices, edges, patches) in the mesh. Then, the least important one is removed and after removal the resulting hole is triangulated. We therefore have a new mesh with a smaller number of triangles.

In our method we started with the vertex decimation proposed by Schroeder[1]. Each vertex is evaluated according to its distance from an average plane (or line) given by its neighbouring vertices, Fig. 1(left).

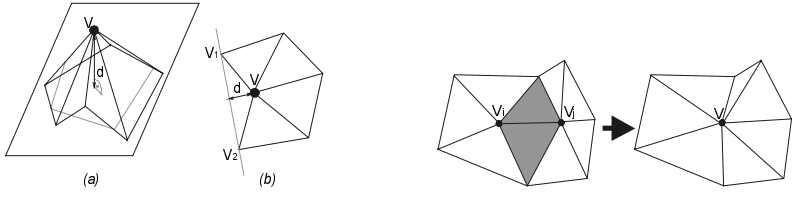

Fig. 1. Vertex importance computation (*left*), edge collapse (*right*)

The second step is to remove the least importance vertex and make a new triangulation. Since we are in 3D, the triangulation is quite complex and time consuming. Considering this, we decided to continue with the edge decimation instead. We evaluated all of the edges going out of the vertex and the least important edge is contracted to zero length. This means that the vertex is moved to the endpoint

of chosen edge, which is to be removed from the mesh together with two adjacent triangles, Fig. 1(right).

This method preserves the mesh topology as well as a subset of original vertices.

Parallel Processing

The idea of progressive vertex removal leads to parallel processing. If we imagine one step of iteration with more than one vertex removal processed by several processors, we get simple and efficient parallelization.

The only condition, which must be maintained, is to obtain only one hole per vertex removal; otherwise the triangulation could increase the program complexity and run time significantly. There is a mechanism called independent set of vertices[4, 12] to carry out such a condition. Two vertices can be in the independent set of vertices if they don't share an edge, as shown in Fig. 2.

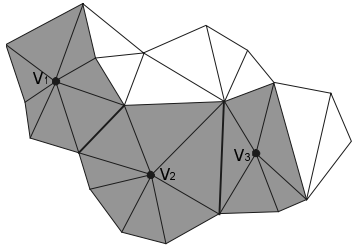

Fig. 2. Independent set of vertices

We assign an importance value to each vertex and then select an independent set to remove vertices with the lowest importance[6]. To construct an independent set from the assigned importance values we process all vertices in order of their importance and select a vertex if none of its neighbours have been taken.

Due to the special data structure, which we used and in order to avoid critical sections in parallel code, we have had to use a stricter rule. We have used a *super* independent set of vertices, where none of two triangles share an edge (vertices v1 and v2 are in the *super* independent on Fig. 2).

Hash Function

Using independent or super independent sets of vertices, we need to sort vertices according to their importance and also to create the independent set of vertices. This appeared to be a critical part of the previous approaches. Since we did not want to use sorting algorithms[7] because of their time complexity, we used a special function to threshold vertices and let only the least few important vertices be considered as candidates for the reduction.

The initial idea was to divide data set (vertices) according to the number of free processors and run decimation as several independent parts. As we already

mentioned, most of data was produced by 3D scanners or iso-surface extraction methods such as Marching Cubes[8]. Considering the principle of both techniques, we can suppose that such triangular mesh is a sequence of strips, where neighbouring vertices are also very close to each other in data file or memory. In other words, if we divide a data set into, let's say, five groups, according to vertices index, there is a good probability that vertices in each group will be close to each other, so the program can run without critical sections except vertices on the *boundary* of each group. Those vertices will not be processed.

This approach brought surprisingly good results for the real object models in the sense of processing time and acceptable quality of approximation. On the other hand there were some problems with the control of simplification degree (instead of 90% reduction you can get 99% as well). Vertices on the border of groups had to be handled in a special way. Another problem arose with artificially generated data or data changed on purpose. Such models don't fit to the assumption about strips and the algorithm is quite ineffective in this case. This experience led us to use a hash function that provides vertex bucketing. This approach enabled us to avoid sorting.

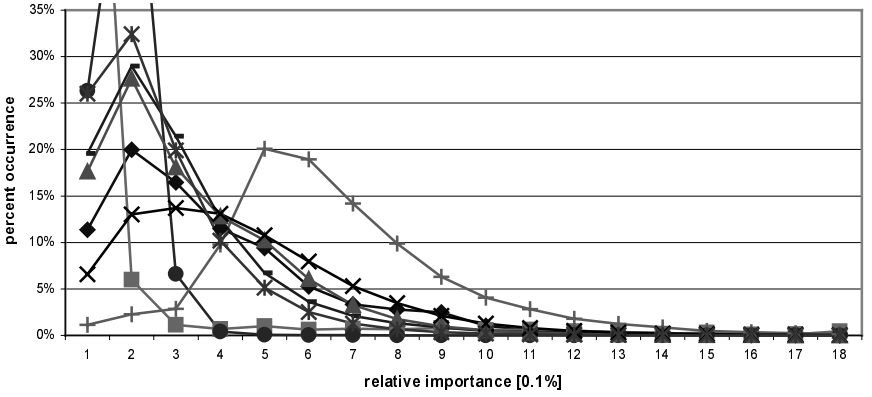

Fig. 3. Vertex importance histogram

New Approach

Instead of vertex sorting or thresholding we can use a hash function and with enough memory we can "sort" vertices in O(N) time complexity, where N is a number of vertices. It is necessary to notice that we do not use modulo operation in the hash function for the address computation to the hash table.

A histogram of vertex importance for tested data sets is shown in Fig. 3. It is obvious that over 80-90% of all vertices have the importance below 1% of maximum importance value.

We used a simple hash function $$y = \frac{x}{|x| + k} \quad \textbf{(1)}$$

shown in Fig. 4 (left).

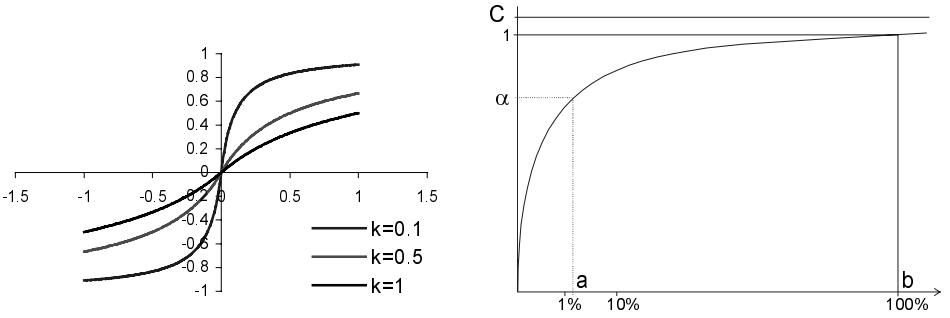

Fig. 4. Graph of the function used (*left*), definition of the hash function (*right*)

This hash function enables us to map the interval $< 0,\infty)$ to $< 0,1)$ non-linearly. Because we need to map only 1% of important vertices the hash function (1) must be modified accordingly, see Figure 4 (right), and scaling coefficient C must be introduced.

$$y = C\frac{x}{x+k} \qquad x \geq 0 \tag{2}$$

From equation (2) we can see that coefficients C and k must be determined somehow. In our approach we decided that we will have two parameters a and α, see Fig. 4 (right), that will be experimentally determined by large data sets processing.

The coefficient b is equal to the maximal importance in the given data set and therefore $f(b)$ must be equal 1. The coefficient a means the boundary for maximal importance of vertices to be considered for processing and α determines the slope of the curve, actually. Those conditions can be used for parameter k and C determination as follows:

$$1 = C\frac{b}{b+k}, \quad b+k \neq 0 \qquad\qquad \alpha = C\frac{a}{a+k}, \quad a+k \neq 0$$

Then

$$C = \frac{b+k}{b} \qquad \text{and} \qquad \alpha = \frac{b+k}{b}\frac{a}{a+k}$$

Solving that we get:

$$k = \frac{ab(1-\alpha)}{\alpha b - a} \qquad\qquad C = \frac{b+k}{b} = 1 + \frac{k}{b}$$

for $b = 1$

$$k = \frac{a(1-\alpha)}{\alpha - a}, \qquad C = 1 + k$$

According to our experiments on large data sets, we have found that the optimal values of coefficients are following: $a = 2$, $\alpha = 80$. This means that 2 percents of the least important vertices will be mapped onto 80% of the whole hash table. The user has to set values of both parameters. The parameter α has a direct influence to the cluster length in the hash data structure, where the cluster length is equal to number of vertices in the same bucket.

Algorithm and Data structure

We described above a framework upon which our algorithm is based. Having described the specific details of the method, we can present our new algorithm now:

1. Evaluate importance of all vertices
2. Make clusters according to the vertex importance
3. Remove vertex from the first cluster, if it is empty continue with the next one
4. Evaluate changed importance of neighbouring vertices and insert vertices in the proper bucket
5. Repeat steps 3 and 4 until desired reduction reached

To make the algorithm more efficient we proposed a special data structure. We use a table of triangles, where we store indexes of all vertices for each triangle. In table of vertices we store each vertex coordinates, a list of triangles sharing this vertex and the address of the cluster where the vertex currently belongs. We also have a vector of clusters where indexes of vertices are stored.

Results

In this section we present results of our experiments that compare achieved time, quality of approximation and also show some examples of reduced models.

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

Table 1. Data sets used

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

Methods Comparison

Fig. 5 shows running time for 96% reduction for three different approaches of vertex ordering. We can see that using the hash function we obtained the best running time in comparison to the other methods. It is necessary to point out that the methods using thresholding or sorting algorithm were implemented in parallel. The run time of hash function is equal to the performance of 2-3 processors running thresholding and it is faster than 8 processors running the method with sort algorithm. It is because both the sort algorithm and the creation of the independent set of vertices were implemented sequentially. With the thresholding we removed the sort algorithm completely, but independent set of vertices remained.

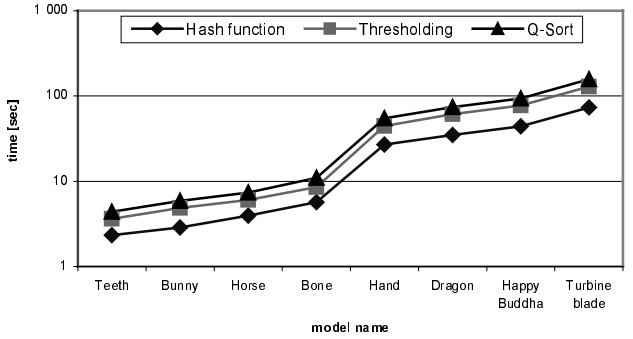

Fig. 5. Achieved time comparison for three mentioned approaches

Unfortunately our results are not directly comparable with other known algorithms, due to the different platforms. 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.

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

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 3. Rough comparison of running times of reduction of the Bunny model.

Algorithm	**Time for a reduction from 69.451 to 1.000 triangles [sec]**
Proposed algorithm	4,1 * η = 4,1* 1,49 = **6,11**
Garland[3]	10,4
Lindstrom & Turk[11]	2585
Hoppe[2]	500
JADE[10]	325

It is obvious that our algorithm is quite fast, however, we do not know the approximation quality reached by the other algorithms. The quality of an approximation can be measured by several approaches. Probably the most popular way is to compute a geometric error using E_{avg} metric[9] derived from Hausdorff distance. As our method keeps the subset of original vertices, we use more simple formula (3):

$$E_{avg}(M_1, M_2) = \frac{1}{k_1} \sum_{v \in X_1} d_v^2(M_2), \quad X_1 \subset P(M_1) \tag{3}$$

where M_1 and M_2 are original and reduced model, 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 set of vertices and reduced model.

If we compare all three above described approaches, we will find that the error values are almost the same. It is also hard to say which method gives the best results, because for each model we get different behaviour of error. Examples of reduced models are presented in Fig. 6,7,8.

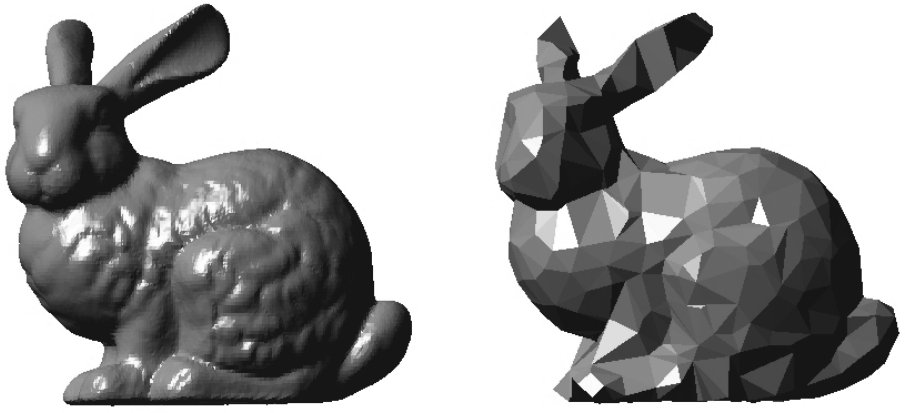

Fig. 6. A bunny model (courtesy GaTech) at different resolutions; the original model with 69,451 triangles on the left, reduced to approx. 1,000 triangles on the right

Fig. 7. A teeth model (courtesy Cyberware) at different resolutions; the original model with 58,328 triangles on the left, reduced to approx. 29,000 triangles in the middle and 6,000 triangles approximation on the right

Fig. 8. The Happyb model (courtesy GaTech) at different resolutions; the original model with 1,087,716 triangles (a), reduced to 105,588 triangles (b), 52,586 triangles (c), 10,974 triangles (d)

Conclusions

We have described our superior algorithm for simplification of triangular meshes, which is capable of producing good approximations of polygonal models.

The algorithm combines Schroeder's decimation (its vertex importance evaluation) with edge contraction to simplify object models in a short time. We have introduced a hash function, which we use instead of expensive vertex sorting. Our algorithm has proved its high speed and simplicity.

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.

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

This work was supported by The Ministry of Education of the Czech Republic: project MSM 235200002

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, 1993.
6. B. Junger, J. Snoeyink. *Selecting Independent Vertices for Terrain Simplification.* In WSCG 98 Proceedings, Pilsen University of West Bohemia, pages 157-164, February 1998.
7. M. Franc, V. Skala. *Triangular Mesh Decimation in Parallel Environment.* In 3rd Eurographics Workshop on Parallel Graphics & Visualization Conference Proceedings, Universitad de Girona, September 2000.
8. W. Lorensen, H. Cline. *Marching Cubes*: *A High Resolution 3D Surface Construction Algorithm.* Computer Graphics (SIGGRAPH 87 Proceedings), 21(4):163-169, July 1987.
9. R. Klein, G. Liebich, W. Straser. *Mesh reduction with error control*. Proceedings of Visualization '96, 1996.
10. A. Ciampalini, P. Cignoni, C. Montani, R. Scopigno. *Multiresolution decimation based on global error*. Technical Report CNUCE: C96021, Istituto per l'Elaborazione dell'Informazione - Condsiglio Nazionale delle Richere, Pisa, ITALY, July 1996.
11. P. Lindstrom, G. Turk. *Fast and memory efficient polygonal simplification.* IEEE Visualization 98 Conference Proceedings, 1998.
12. A.W.F. Lee, W. Sweldens, P. Schroder, L. Cowsar. *MAPS: Multiresolution Adaptive Parameterization of Surfaces*. In SIGGRAPH '98 Proceedings. 1998.