Proceedings of ALGORITMY 2000
Conference on Scientific Computing, pp. 357 - 367

PARALLEL TRIANGULAR MESH REDUCTION

MARTIN FRANC[1], VÁCLAV SKALA[2]

Abstract. The visualization of large and complex models is required frequently. This is followed by number of operations which must be done before visualization itself, whether it is an analysis of input data or a model simplification. One of the techniques that enhance the computational power is parallel computation. It can be seen that multiprocessor computers are more often available even for ordinary users. Together with Microsoft Windows expansion we have easy and comfortable tools for multiprocessor (multithread) programming as well. We present an original efficient and stable algorithm for triangle mesh simplification in parallel environment. We use a method based on our original super independent set of vertices to avoid critical sections. Programs have been verified on MS Windows platform using standard Borland Delphi classes for multithread programming.

Keywords: data visualization, triangular mesh reduction, algorithm complexity, computer graphics, parallel programming.

1. Introduction. Simplification of large complex models is a common task in visualization. The simplification of models finds its use in virtual reality, whenever the object is in large distance from the observer or when details of the model are irrelevant or we just need rough view for fast manipulation. We show that the efficiency of our algorithm is independent on a chosen simplification technique. Our algorithm works with a *super* independent set of vertices [1] for vertex elimination to avoid critical sections in program code, which normally decrease the speed of computation. As a programming tool we use standard Borland Delphi tools together with a TThread class, which encapsulates attributes and methods given by MS Windows for multithread programming.

In section 2 of this paper we describe the simplification in general and techniques we use in our algorithm. Section 3 introduces tools for multithread programming under MS Windows. This section is followed by section number 4, where we discus general problems of shared memory and explain *super independent set* term. In section 5 we present our algorithm and section 6 shows achieved results.

2. Popular techniques. Decimation methods are simplification algorithms that start with a polygonization (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.

Department of Computer Science, University of West Bohemia, Univerzitní 8, Box 314, 306 14 Plzeň, Czech Republic, (e-mail**: marty@students.zcu.cz** and **skala@kiv.zcu.cz)**
http://iason.zcu.cz/~{marty | skala }

[1] Was supported by the Ministry of Education of the Czech Republic - project VS 97155
[2] Was supported by the Academy of Sciences of the Czech Republic - project A2030801

VERTEX DECIMATION METHODS

One of the most used method is vertex decimation, an iterative simplification algorithm originally proposed by Schoreder [2]. In each step of decimation process, a vertex is selected for removal. All the facets adjacent to that vertex are removed from the model and the resulting hole is retriangulated. Each vertex is evaluated by its importance in the mesh. The vertex with low importance is eliminated. In general, there are plenty of techniques simplifying triangular meshes by vertex elimination. The difference among them is the way of the vertex importance evaluation and kind of retriangulation. Since the retriangulation 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.

EDGE DECIMATION

Other subset of decimation techniques is pointed to the elimination of the whole edge. When an edge is contracted, a single vertex replaces its endpoint. Triangles, which degenerate to an edge, are removed. Hoppe [6] appear to have to be the first who used edge contraction as the fundamental mechanism accomplishing surface simplification. It is necessary to evaluate the importance of edges before the contraction. One of the best-known technique [3] uses *quadric error metrics* for the edge (vertex pairs) evaluation. The edges are contracted according to their importance – the less important edges first, similarly to the case of vertex removal. Unless the topology is explicitly preserved, edge contraction algorithms may implicitly alter the topology by closing holes in the surface.

PATCH (TRIANGLE) DECIMATION METHODS

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

OUR APPROACH – FRAMEWORK OF OUR ALGORITHM

Each of the above mentioned approaches have their advantages and disadvantages [8]. We have tried to extract the advantages of all approaches as will be presented in the following part. We started with vertex decimation methods and used the Schroeder's approach 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 (discussed above). Instead the vertex elimination and arising hole retriangulation, one of adjacent edge can be contracted as well. Removing a vertex by edge contraction is generally more robust than projection of neighbourhood onto a plane a retriangulation. In this case, we do not need to worry about finding a plane onto which the neighbourhood can be projected without overlap.

Firstly, each vertex in a mesh is evaluated according to its importance. Then the one with the lowest importance is marked and the most suitable edge for the contraction is searched in its neighbourhood. As the most suitable edge for contraction we take the one, which goes out of the eliminated vertex, that does not cause the mesh to fold over itself, and is best preserving the original surface according to our criterion. As the criterion we use either the contraction of the shortest edge, or the criterion of minimal retriangulated area. To prove the method independence of our algorithm, we have tested some more heuristic based on vertex decimation, besides Schroeder's method. These heuristics are described in [5] in detail and their comparison can be found in section 5 of this paper.

Since the contraction can potentially introduce undesirable inconsistencies or degeneracies into the mesh, we must apply some consistency checks to a proposed contraction. If one of the checks fails, we discard the contraction and use another edges if any still remains.

3. Multithread programming . We developed the algorithm for the Windows NT platform, using Borland Delphi. There are quite easy and efficient tools for multithread programming.

A thread is an operation system object, where program code is run. For every application at least one (*primary*) thread is created. Each thread can create other new threads, during its run. These threads share the same address area and can perform either the same or different action. After the *primary thread* is finished (together with all its threads), the application is terminated and the process is erased from the system. Threads allow all program routines to run all at once. If there in one CPU only, threads alternate (so called preemptive multitasking), otherwise they run concurrently.

Threads can be used to improve application performance by managing input from several communication devices, or distinguishing among tasks of varying priority. For example, a high priority thread handles time critical tasks, and a low priority thread performs other tasks. In Borland Delphi, there is a standard class named *TThread*, which encapsulates all attributes and methods for multithread programming that MS Windows allows.

4. Super independent set.

INDEPENDENT SET

A basic idea, which has been used in theoretical work [4] recently, is that decimation by deleting an independent set of vertices (no two of which are joined by an edge) can be run efficiently in parallel. The vertex removals are independent and they leave one hole per one deleted vertex, which can be retriangulated independently. This decreases the program complexity and run time significantly. Since deletion and retriangulation is related to the degree of vertices being removed (in the worst case with O (d^2) time complexity, where d is the vertex degree), Kirkpatrick[4] has advocated deleting low degree vertices ($d < 10$) and proved that this still allows large independent sets (>1/6 of all vertices). However, this approach ignores the preservation of the model shape. Therefore we use a technique [5] when we assign an importance value to each vertex, then select an independent set to delete by greedily choosing vertices of low importance relative to their neighbours.

It is natural to use a greedy strategy to construct an independent set from an assignment of importance values. It means to go through all the vertices in order of their importance and take a vertex if none of its neighbours have been taken. It means that only those vertices that do not share an edge with the each other can be in the independent set.

INDEPENDENT SET – WITHOUT NEED OF CRITICAL SECTIONS

We have developed and use a *super independent* set, where every two triangles including two independent vertices can not share an edge, see Figure 1.

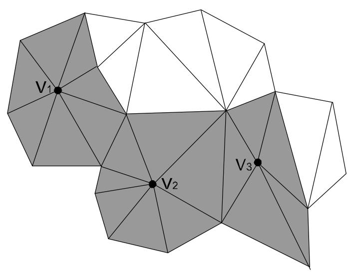


FIG. 1*: Vertices v_1, v_2, v_3 are independent to each other, but only the vertices v_1 and v_3 are super independent.*

If we remove one vertex from the independent set, the removal change the properties of the vertex neighbours. That affects neighbourhood of other vertices in the set. Even vertex neighbours are independent in this *super independent* set, so vertices are completely independent and the parallelization can be done without critical sections in program code. Due to the data structures used we can create the independent set in *O (n)* time, where *n* is the number of vertices.

5. Parallel Algorithm. Our new parallel algorithm can be described as:

1. Divide the set of vertices into *N* parts; *N* is equal to the number of free processors.
   - Get the number of processors.
   - Divide the set of vertices into *N* parts of the same number of vertices.
2. Run *N* threads to evaluate vertex importance according to its topology. Each thread makes a computation on its own set of vertices.
   - Determine a vertex topology.
   - For *simple*, *boundary or interior edge* vertices, compute their importance. The importance for any other type of vertex is set to any high value.

3. After all threads finish their task, sort (Quick Sort algorithm) all the vertices according to their importance in increasing order.
4. Find a super independent set of vertices. For each vertex do:
   - If the vertex is marked as unused in the independent set (initially all vertices are marked as unused), check its neighbours. If all the neighbours are unused, put the vertex into the independent set and mark the vertex and all its neighbour vertices as used.
   - Used vertices and their neighbours are skipped.
5. Divide the independent set of vertices into N parts.
6. Run N threads for decimation. Each thread makes the decimation on its own set of vertices.
   - For each eliminated vertex, find the optimal edge for contraction that includes it.
   - Test the consistency of the mesh if this edge is contracted (removed).
   - If the consistency test is OK, remove the vertex and retriangulate the arising hole, otherwise find another short edge and go to the previous point.
7. Repeat steps 1– 6 until the required degree of the mesh reduction is reached.

6. Experimental results. In this section we present results of our experiments that compare an acceleration and efficiency obtained with different data sets, using different number of processors (threads).
We use 6 different data sets for the experiments, see Table 1.

Model name	No. of triangles	No. of vertices
Horse	96,966	48,485
Bone	137,072	60,537
Bell	426,572	213,373
Hand	654,666	327,323
Dragon	871,414	437,645
Happyb	1,087,716	543,652

TABLE 1: Data sets used.

Table 2 shows time comparison achieved by reducing the models using 1 to 8 – processor computer (DELL Power Edge 8450 – 8xPentium III, cache 2MB, 550MHz, 2GB RAM, running on the Windows 2000).

Model name	Time [sec] obtained with different number of processors (threads) used							
	1	2	3	4	5	6	7	8
Horse	8.9	6.3	5.5	4.9	4.7	4.5	4.4	4.3
Bone	13.5	9.4	8.2	7.6	7.0	6.8	6.6	6.4
Bell	62.7	48.1	42.4	39.4	37.8	36.9	35.4	34.6
Hand	69.2	51.5	44.8	41.0	39.7	38.4	37.5	36.7
Dragon	93.6	69.5	61.5	57.6	54.3	52.6	51.3	50.6
Happyb	118.3	89.1	78.1	73.2	69.3	67.8	65.9	64.5

TABLE 2: *Obtained time (in seconds) for 90% reduction on 1 to 8 processors active.*

We investigated the acceleration and the efficiency for different size of data sets according to the number of processors used.

ACCELERATION COMPARISON

The acceleration *a* is computed from total times (sequential and parallel parts of the algorithm together) using expression

$$a = \frac{time_N}{time_1}, \tag{1}$$

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

Figure 2 shows the acceleration of individual sequential and parallel part of the algorithm for the *Happyb* model. Total time means run time of the whole algorithm, Imp. eval. is time for importance evaluation, Qsort is sorting time, Iset is time for selecting the super independent set and Decim is time of the decimation part.

EFFICIENCY COMPARISON

The efficiency e is defined as follows:

$$e = \frac{time_N}{N * time_1}, \tag{2}$$

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

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 (3) perfectly.

$$a_{teor} = \frac{1}{(1-p) + \dfrac{p}{N}}, \qquad p = \frac{N*(1-a_{teor})}{a_{teor}*(N-1)}, \qquad (3), (4)$$

where p is potentially parallel code and N is the number of processors used.

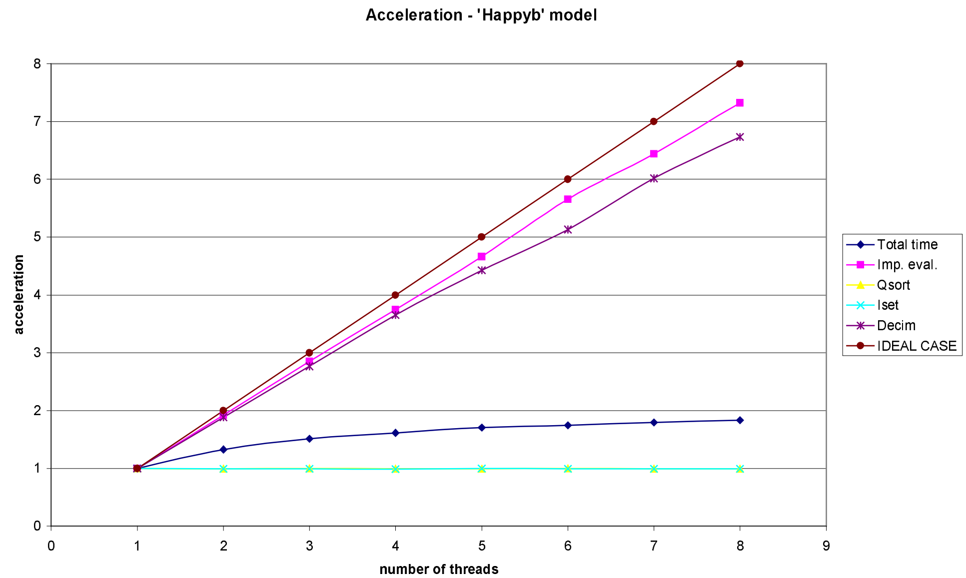


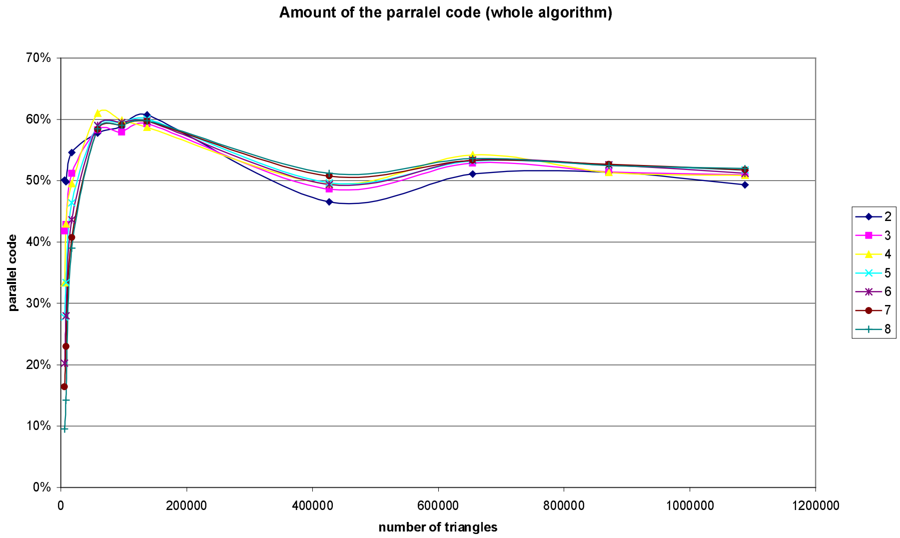
FIG. *2: The acceleration of individual sequential and parallel part of the algorithm for the Happyb model.*

The value of potentially parallel code is independent from the number of processors used, see Table 3, and for the large model Happyb the value p = 0.51 was reached for the whole algorithm.

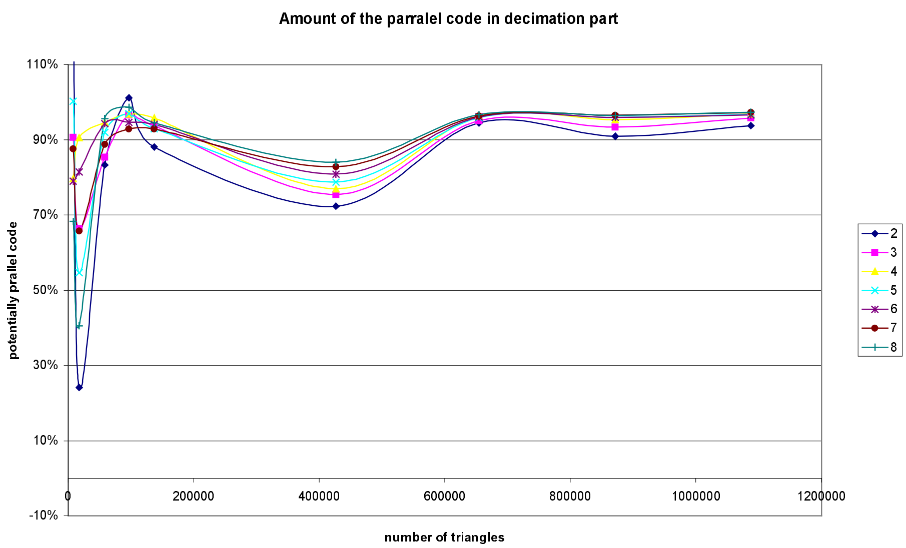
	Number of processors (threads) used							
	1	2	3	4	5	6	7	8
e	1	0.66	0.5	0.4	0.34	0.29	0.25	0.22
a	1	1.32	1.51	1.61	1.7	1.74	1.79	1.83
a_{teor}	**1**	1.32	1.51	1.61	1.7	1.74	1.79	1.83
p	**X**	0.49	0.51	0.50	0.51	0.51	0.51	0.52

TABLE 3: *The experimental results and theoretical calculations according to Amdahl's law; computed for the happyb model.*

Figure 3 shows the amount of potentially parallel code according to the number of triangles, for different number of processors used**.**



(a)



(b)

FIG. 3: *The amount of the parallel code for the whole algorithm (a) – the potentially parallel code according to Amdahl's law is approx. 51%, the amount of the parallel code for the decimation part (b).*

TIME COMPARISON

The work itself was inspired by recent work in this field, especially by Seidel [7]. Unfortunately the results are not comparable directly due to the different platforms. To make the results roughly comparable at least, we use the official benchmarks presented by SPEC as shows table 4, where η presents the superiority of DELL computer against the SGI. Table 5 presents our results according to results obtained recently [7] taking the ratio η into the consideration. Our algorithm is 2.72 times faster according to the table 5.

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 4: *Benchmark test presented by Standard Performance Evaluation Corporation*

Model name	Time [sec] of 99.15% reduction		ratio		
	SGI R10000 [7]	DELL 8450 [1 thread]	μ (SGI/DELL)	η	μ / η
Dragon	584.9	123.3	4.74	1.74	2.72

TABLE 5: *Rough comparison*

INDEPENDENCE ON DECIMATION METHOD

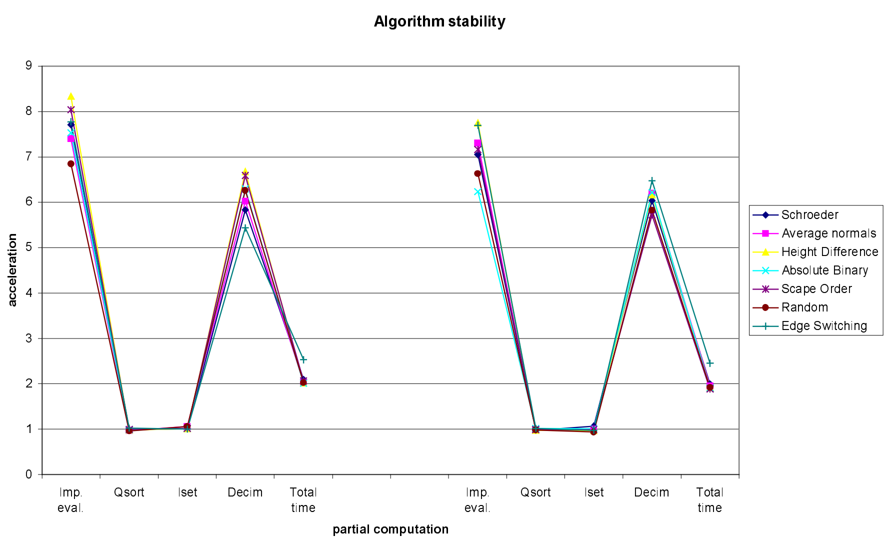


FIG. 4*: The acceleration of partial parts of the algorithm for several vertex importance evaluation heuristics. Criterion of minimal area retriangulation on the left side, the shortest edge contraction criterion on the right side of the graph.*

As we have already mentioned, we use some more vertex decimation heuristics to test the independence of our algorithm. In Figure 4, there is a graph of the acceleration comparison according to partial parts of algorithm (computation) for 7 heuristics. On the left side, there is a graph of decimation according to the criterion of minimal area after the retriangulation, on the right there is a graph of decimation according to the shortest edge contraction criterion, see chapter 2.2. The heuristics are intimately described in [5]. They are based on the same principle, except the method of their vertex importance evaluation. The following methods have been used: Schroeder‘s method, Average Normals, Height difference, Absolute Binary, Scape Order, Random (any of previous five) and Edge Switching.

It is obvious that the algorithm is independent of the heuristic used in meaning of the efficiency and the acceleration.

PROGRAM OUTPUT

Figure 5 shows examples of original and reduced models.

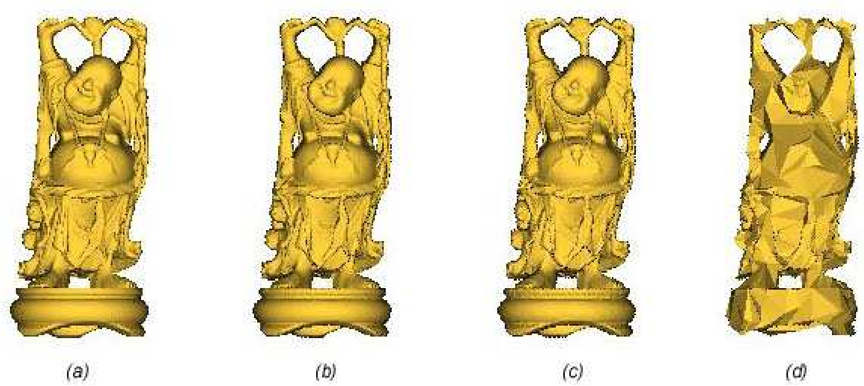


FIG. 5: *The Happyb model (courtesy GaTech) at different resolutions; the original model 1.087.716 triangles (a), reduced to 105.588 triangles (b), 52.586 triangles (c), 13.100 triangles (d).*

Conclusion. We have described the new original algorithm for triangular mesh simplification with its parallel modification. The algorithm combines vertex decimation method with the edge contraction to simplify object models in a short time. We have used the super independent set of vertices to improve the parallelization. It enables us to make fast parallel algorithm without use of any synchronization technique such as critical sections, so the system overhead is minimal. Our experiment proved that we reached high effectivity of parallelization $p = 0.87$ (if the overhead cost by the TThread class is included) or $p = 0.97$ (if the overhead cost by the TThread class is not included) for parallelized parts. This also proved that high level constructions and object oriented programming can be used at the application level with high effeciency guarantee. The proposed method proved its stability according to the number of processors and the size of the data set used.

Acknowledgements. We would like to express our thanks to DELL Computer Czech Republic for enabling us to carry out all the experiments on their 8-processor computer type, DELL Power Edge 8450 – 8xPentium III, cache 2MB, 550MHz, 2GB RAM. We also benefited from the large model repository located at Georgia Institute of Technology;

URL**:** **http://www.cc.gatech.edu/projects/large_models**.

REFERENCES

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

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

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

[4] D. Kirkpatrick. *Optimal Search in Planar Subdivisions.* SIAM J. Comp., pages 12:28-35, 1993

[5] B. Junger, J. Snoeyink. *Selecting Independent Vertices for Terrain Simplification.* In WSCG 98 Proc., Pilsen University of West Bohemia, pages 157-164, February 1998

[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. *Mesh optimization.* In SIGGRAPH 93 Conference Proc. pages 19-26, 1993

[7] H. P. Seidel, S. Campagna, L. Kobbelt, R. Schneider, J. Vorsatz. *Mesh Reduction and Interactive Multiresolution Modeling on Arbitrary Triangle Meshes.* In SCCG 99 Proceedings, Comenius University Bratislava, pages 34-44, 1999

[8] M. Garland. Multiresolution Modeling: Survey & Future Opportunities. In the SIGGRAPH 97 course notes, 1997