Fast Mesh Simplification Algorithm Based on Edge Collapse

Shixiang Jia[1], Xinting Tang[2], and Hui Pan[3]

[1] Department of Computer Science and Technology, Ludong University,
264025 Yantai, P.R. China
jiashixiang@gmail.com
[2] Department of Computer Science and Technology, Ludong University,
264025 Yantai, P.R. China
jsjtang@hotmail.com
[3] Department of Computer Science and Technology, Ludong University,
264025 Yantai, P.R. China
panhui@gmail.com

Abstract. Firstly, we present a new mesh simplification algorithm. The algorithm is based on iterative half-edge contracting, and exploits a new method to measure the cost of collapse which takes the length of contracting edge and the dihedral angles between related triangles into account. The simplification does not introduce new vertex in original mesh, and enables the construction of nested hierarchies on unstructured mesh. In addition, the proposed algorithm adopts the Multiple-Choice approach to find the simplification sequence, which leads to a significant speedup with reduced memory overhead. Then we implement a mesh simplification system based on this algorithm, and demonstrate the effectiveness of our algorithm on various models.

1 Introduction

Many high-resolution models are obtained by scanning systems or created by modeling systems. Unfortunately, these highly detailed models are hard to store and transmit, and will slow down the rendering speed, causing jerkiness of movement. In fact, such complex models are not always required. In order to get simpler versions of them, a simplification algorithm is needed. As for the algorithm, it takes a complex model and automatically generates an approximation using fewer triangles that looks reasonably similar to the original.

We provide a new algorithm which tries to preserve the visually important parts of the model by using a new cost function to measure the approximation error. As the visual acuity of the human vision system is principally dependent upon three factors: size, orientation and contrast [1,2], our new cost function will focus on the edge length and the dihedral angles between related triangles. In order to speed up the algorithm, we use a probabilistic optimization strategy based on the Multiple-Choice Algorithm [3] to find the optimal decimation sequence. The Multiple-Choice technique does not require a global priority queue data structure which reduces the memory overhead and simplifies the algorithmic structure. We have developed a simplification system based on the proposed algorithm, and applied our algorithm on many models with various size.

The rest of the paper is organized as follows. We first review the related work in Section2. Section 3 describes our algorithm in detail. The implementation is discussed in Section 4. Section 5 presents a discussion of results and performance analysis. Section 6 concludes the paper.

2 Related Work

The problem of surface simplification has been studied in both the computational geometry and computer graphics literature for several years. Some of the earlier work by Turk [4] and Schroeder [5] employed heuristics based on curvature to determine which parts of the surface to simplify to achieve a model with the desired polygon count. Vertex clustering algorithm described by Rossignac and Borrel [6] is capable of processing arbitrary polygonal input. A bounding box is placed around the original model and divided into a grid. Within each cell, the cell's vertices are clustered together into a single new representative vertex. The method is very fast and effective, however, the quality of the approximation is not often satisfactory. This approach usually leads to a vertex distribution which does not adapt to the local curvature of the surface, and can not guarantee a proper manifold topology of the resulting approximation.

Hoppe [7,8] posed the model simplification problem into a global optimization framework, minimizing the least squares error from a set of point-samples on the original surface. Later Hoppe extended this framework to handle other scalar attributes, explicitly recognizing the distinction between smooth gradients and sharp discontinuities. He also introduced the progressive mesh [8], which is essentially a stored sequence of simplification operation, allowing quick construction of any desired level of detail along the continuum of simplifications. However, the algorithm provides no guaranteed error bounds.

There is considerable literature on surface simplification using error bounds. Cohen and Varsheny [9] have used envelopes to preserve the model topology and obtain tight error bounds for a simple simplification. An elegant solution to the polygon simplification problem has been presented in [10,11] where arbitrary polygonal meshed are first subdivided into patches with subdivision connectivity and then multiresolution wavelet analysis is used over each patch. These methods preserve global topology, give error bounds on the simplified object and provide a mapping between levels of detail.

Garland [12] used iterative contractions of vertex pairs to simplify models and maintains surface error approximation of polygonal modes. This algorithm is efficient and can rapidly produce high quality approximation. Incremental decimation algorithms typically lead to superior model quality. These algorithms simply models by iteratively executing atomic decimation step such as edge collapse (see Fig. 1). An edge collapse takes the two endpoints of the target edge, moves them to the same position, links all the incident edges to one of the vertices, deletes the other vertex, and removes the faces that have degenerated into lines or points. Typically, this removes two triangular faces per edge contraction. To minimize the approximation error, a cost function measuring the quality of the approximation is proposed to guide the process of simplification [7,12].

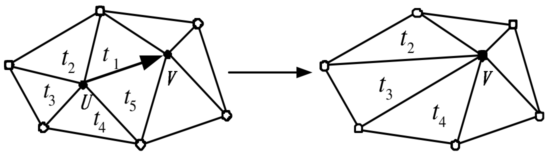


Fig. 1. Half-edge collapse. The (*u, v*) edge is contracted into point v. The t_1 and t_5 triangles become degenerate and are removed.

The particular sequence of edge collapse transformations must be chosen carefully, since it determines the quality of the approximating models. For those algorithms [7], the priority queue is a natural data structure to store the order of the edges to be simplified, which allows a variety of operations (inclusion, access and removal of the largest, etc.) to be efficiently performed. But it takes a long time to build the queue before starting the simplification process. Furthermore, each step of the decimation also consumes a significant amount of time to recompute the collapse cost of changed edges and to update their position in the priority queue. In order to accelerate this process, Wu and Kobbelt [3] have presented a technique called Multiple-Choice based on probabilistic optimization. It makes no use of a priority queue, but chooses the edge to be contracted from a small number of randomly selected edges.

We provide a new algorithm which can preserve the visually important parts of the model by using a new cost function to measure the approximation error. In order to speed up the algorithm, we also use a probabilistic optimization strategy based on the Multiple-Choice Algorithm to find the optimal decimation sequence. Our system allows faster simplification than some quality method.

3 Simplification Algorithm

3.1 Atomic Decimation Operator

Our algorithm is based on the half-edge collapse operation. Half-edge collapse is to choose an edge (*u*,*v*) and contract it to one of its endpoint *v*. After collapsing, all triangles adjacent to either *u* or *v* are connected to *v*, and triangles adjacent to both *u* and *v* are removed (see Fig. 1).

We prefer half-edge collapse because of its simplicity. The methodology of half-edge collapse is in fact closely related to the vertex decimation approach. In each step of vertex decimation approach, a vertex is selected for removal. All the facets adjacent to that vertex are removed from the model and the resulting hole is triangulated. Instead of the vertex elimination and arising hole triangulation, half-edge contracting just merge one endpoint of the selected edge into the other endpoint. Half-edge contracting avoids the action of hole triangulation, and is generally more robust than vertex decimation. In this case, we do not need to worry about finding a plane onto which the neighborhood can be projected without overlap. In addition, half-edge contracting makes progressive transmission more efficient (no intermediate vertex coordinates) and enables the construction of nested hierarchies that can facilitate further applications.

The change caused by edge collapse is quantified by collapse cost. The algorithm based on edge collapse has to solve two problems: one is how to calculate the collapse cost for every candidate; the other is how to find the simplification sequence. Then the algorithm can collapse the edge iteratively until the given criterion is satisfied.

3.2 Collapse Cost

According to the characteristics of human vision system, observers are mainly sensitive with three attributes of the model: size, orientation and contrast [1,2]. According to the first attribute, the length of the edge should be considered when calculating its collapse cost. With the last two attributes, the dihedral angles between the related triangles are also important guidance. Our cost function will focus on the edge length and the sum of the dihedral angles.

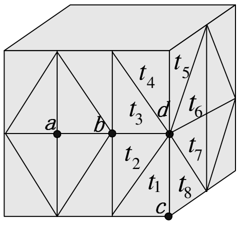


Fig. 2. The candidate for contracting

The principle of our algorithm is that the contracting edge should be at smooth areas (such as edge(a,b) in Fig. 2), so the dihedral angle between any two related triangles should be small. To calculate the collapse cost for edge(u,v) in Fig. 1, we need to do some work as follows:

1) Find out all the triangles adjacent to vertex u: t_1, t_2, t_3, t_4, and t_5, and those adjacent to both vertex u and v: t_1 and t_5

2) Calculate the dihedral angle between t_1 and t_2, t_3, t_4, t_5, and then those between t_5 and t_1, t_2, t_3, t_4

3) Set the largest dihedral angle in step 2 as the final angle between the related triangles adjacent to edge(u,v).

The final angle of edge(a,b) in Fig. 2 is very small(zero), so we can contract it. As a matter of fact, we can relax this condition to that when the edge is an exclusive edge we can also collapse it, such as edge(c,d) in Fig. 2. The collapse of edge(c,d) will have little influence to the appearance of the model. We can observe that the dihedral

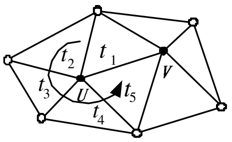


Fig. 3. The calculation of the triangles' weight

angle between t_1 and t_8 is very large while the one between t_1 and t_2 is very small. If we use the above algorithm for calculation, the collapse cost of edge(c,d) will be large, which is contrary to the fact, so we need to improve it.

We give every triangle a weight when calculating the dihedral angle. For edge(u,v) in Fig. 3, when calculating the dihedral angle between t_1 and the other triangles, we think the one between t_1 and t_2 is most important , so the weight of t_2 to t_1 should be largest, and the weights of t_2, t_3, t_4 and t_5 should decrease counterclockwise. While, when we calculate the dihedral angle between t_5 and the other triangles, the weights of t_4, t_3,, t_2 and t_1 should decrease clockwise.

We define that S is the set of triangles that are adjacent to vertex u, the number of the triangles in it is n and s_i (i=1, 2 , , ,n) indicates the ith triangle. B is the set of triangles that are adjacent to both u and v, the number of the triangles in it is m. We define the weight of s_i to b_j as follows:

$$W(s_i, b_j) = n/(n + D(s_i, b_j)) \tag{1}$$

where $D(s_i, b_j)$ in (1) denotes the number of triangles between s_i and b_j. In Fig. 3, if b_j is t_1, $D(s_i., t_1)$ denotes the number of triangles which will be visited when traversing counterclockwise from t_1 to s_i. For example, $D(t_2, t_1)$=1, $D(t_4, t_1)$=3. If b_j is t_5, $D(s_i, t_5)$ denotes the number of triangles which will be visited when traversing clockwise from t_5 to s_i.

Define f_i (i=1, 2, , , n) indicates the unit normal vector of the ith triangle of S, and e_j (j=1, 2, , , m) indicates the unit normal vector of the jth triangle of B. We define the collapse cost of edge (u, v) is:

$$Cost(u,v) = \| u - v \| \times (\sum_{j=1}^{m} \sum_{i=1}^{n} [(1 - (e_j \bullet f_i)) \times W(s_i, b_j)]) \tag{2}$$

where $\|u\text{-}v\|$ in (2) indicates the length of edge(u,v).

$$e_j \bullet f_i = |e_j| \times |f_i| \times \cos\theta = \cos\theta \tag{3}$$

We use $e_j \cdot f_i$ to compare the value of the dihedral angleθ, so we can avoid the calculation of arccosine.

3.3 Multiple-Choice Algorithm

Since the cost function has been defined, each possible atomic decimation operation (candidate) can be rated according to the function. So the remaining issue is to choose a candidate for each decimation step. In other words, we should find the optimal decimation sequence. Finding the optimal decimation sequence is a very complex problem [13] and consequently one has to find solutions with approximate optimality. Most of the algorithms adopt a greedy strategy to find a decimation sequence that is close to the optimal. For every decimation step, the algorithm will go through all possible candidates to find one with the lowest cost. An implementation of the greedy strategy usually requires a priority queue data structure for the candidates that has to be initialized and updated during the decimation.

Our algorithm uses a different probabilistic optimization strategy based on Multiple-Choice algorithm to find the decimation sequence. The fundamental idea behind

MCA is quite simple and intuitive and can be explained best by means of the well-established bins-and balls model [14,15].

In order to apply MCA to the model simplification problem we have to map balls, bins, and maximum load to the corresponding mesh entities [3]. Since the balls are enumerated in the outer loop (for each ball make a MC decision) they correspond to the decimation steps. The bins represent the possible choices in each step, hence they correspond to the possible candidates. The maximum load finally is the value that is to be optimized and consequently we associate it with the quality criterion that is used to rate the candidates. In this setup, the MCA approach to model simplification consists of testing a small set of d randomly selected candidates (edge collapses) in each step and performing that decimation operation among this small set that has the best quality value. Experiments show that using MCA approach our algorithm can produce approximations in almost the same quality as other algorithms based on greedy strategy when $d = 6$.

Compared to the greedy optimization, the major benefit of the Multiple-Choice optimization is that the algorithmic structure is much simpler. For the Multiple-Choice optimization we do not need a priority queue and consequently we reduce the memory consumption and make the algorithm much easier to implement.

3.3 Algorithm Summary

Firstly, the importance of each vertex in a mesh should be evaluated. The most suitable edge for the contraction is searched in its neighborhood, and the one with the lowest cost is marked as the vertex's importance. As for the most suitable edge for contraction we take the one that does not cause the mesh to fold over itself and preserves the original surface according to the criterion. Then we can decimate vertices one by one according their importance.

Using the above idea of Multi-Choice techniques, the overall framework of our algorithm can be summarized as follows:

1. Determine the topology structure of the original mesh and calculate the unit normal of every triangle in the mesh.
2. For every vertex of the original model, calculate the cost of contracting the vertex to its neighborhood, which means to calculate the cost of all the edge adjacent to the vertex, picking the edge with the lowest cost as the vertex's collapse edge.
3. Randomly choose d vertices from all candidates, and update the vertices needed to be recomputed among the d vertices.
4. Select the vertex with candidature edge of lowest cost from the d vertices, and contract its edge. After contracting, mark the related vertices needed to be updated.
5. Repeat step 3 and 4 until the given criterion is satisfied.

Our algorithm does not need a global priority queue, so it is much easier to be implemented. In Table 1, we compare our algorithm with others.

Table 1. Compared with others, our algorithm based on greedy strategy does not need a global priority queue

Algorithm step	Our algorithm	Others
Initialize	Initialize, compute collapse cost for all candidate	Initialize, compute collapse cost for all candidate, perform global queue sorting
Select candidate	Select *d* vertices randomly, update the vertex's cost if necessary, pick the best out of *d*	Top of the queue
Decimate	Perform operator	Perform operator, locally recomputed cost, update global queue

4 Implementation

Based on the above algorithm, we have developed a framework providing efficient simplification on models of various size.

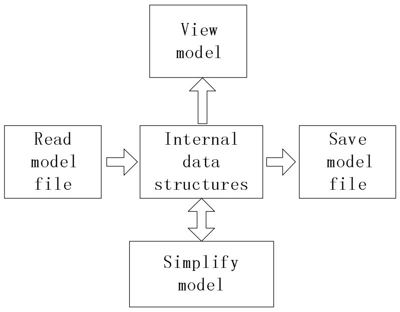


Fig. 4. The structure of our simplification system. The system can read model file into internal data structure, simplify it interactively, and save the internal data back to model file.

The processing stage of the framework consists of the following steps (see Fig. 4):

1. Read the input model file, and create internal data structure.
2. Render the current model.
3. Simplify the model according to the user's aim.
4. Render the simplified model.
5. Repeat step 2-4 until the user is satisfied with the resulting model, then save it back to model file.

Step1 consists of reading the model file, triangulating the model, and storing all the points and triangles into the internal data structures. As our simplification algorithm can only deal with triangles as input, the models consisting of n-sided polygons need to be triangulated in the preprocessing phase.

```
Class Vertex{
Vec3   location; // Vec3 is a vector class
int     index;
set<int> vertNeighbors;
//vertices connected to this vertex
set<int> triNeighbors;
//triangles of which this vertex is a part
bool   bActive;
//false if vertex has been removed
double cost;
//cost of contracting the min cost edge
int  minCostNeighbor;
// index of vertex at other end of the min cost edge
}
Class Face{
float weight;
Vec3  direct;
Vec3  normal;//normal of this triangle
Vec3  point;// vertices of this triangle
bool  bActive; // active flag
void  getVerts(int& v1,int& v2,int& v3)
}
Class CShape
{
vector <Vertex> vertices;
vector < Face > faces;
Vertex& getvertex (int i);
{ return vertices( i ) };
Face& getface (int i ) { return faces( i );};
unsigned int vert_count( ) const;
unsigned int face_count( ) const;
unsigned int active_vert_count( );
unsigned int active_face_count( );
bool initialize( );
//find min cost edge for every vertex
}
Class CModel
{
vector <CShape> shapes;
void getshape (CShape& shape);
bool initialize( ); //initialize every shape
bool decimate(int percent);
//control the process of simplification
bool EdgeCollapse( int vertid )//contracting one edge
}
```

As shown above, we define the basic internal data structure for model and simplification with the aid of Visual C++.

5 Results and Discussion

In this section the efficiency of our algorithm is demonstrated. We have tried our implementation on various models of different sizes and varying shapes, and have achieved encouraging results. Table 2 summarizes the running time of our current implementation and compares it with Garland's QEM algorithm [12] and Melax's algorithm which is simple and fast [16]. All experiments are done on a commodity PC with Intel 2.4GMHz CPU and 1024M RAM.

Table 2. Running time of different algorithms. All data reflects the time needed to simplify the model to 0 triangles

Model	Ver-tices	Trian-gles	Algo-rithm	Initialization time (secs.)	Simplification time (secs.)	Total (secs.)
tele-phone	34352	68575	QEM	3.721	19.493	23.214
			Melax	3.058	25.143	28.201
			Our	4.344	13.085	17.429
lamp	5848	11672	QEM	1.245	3.506	4.751
			Melax	2.215	5.278	7.493
			Our	1.687	2.110	3.797
bunny	34834	69451	QEM	5.512	19.970	25.482
			Melax	8.531	25.225	33.756
			Our	5.122	6.965	12.087

We also depict the absolute maximum geometric errors for the bunny and lamp model when decimating them to various levels of details (see Fig. 5 and Fig. 6). The approximation error is measured by the Hausdorff distance between the original model and the simplified result. The Hausdorff distance (sometimes called the L^{∞} norm difference) between two input meshes M^1 and M^2 is [17]:

$$K_{haus}(M^1,M^2) = \max(dev(M^1,M^2), dev((M^2,M^1)) \tag{4}$$

where $dev(M^1, M^2)$ in (4) measures the deviation of mesh M^1 from mesh M^2. The Hausdorff distance provides a maximal geometric deviation between two meshes.

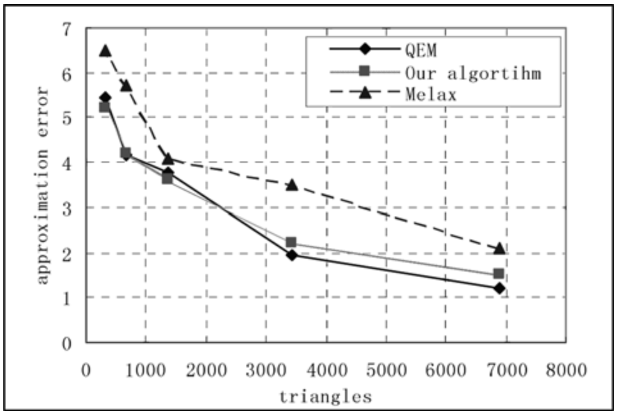


Fig. 5. Absolute maximum geometric error for bunny model. The size of bounding box is 15.6*15.4*12.1.

Fig.7 demonstrates the visual quality of the approximations generated using our algorithm. In Fig.7 (i), the bunny model is drastically simplified (99%), but the major details of the original still remain.

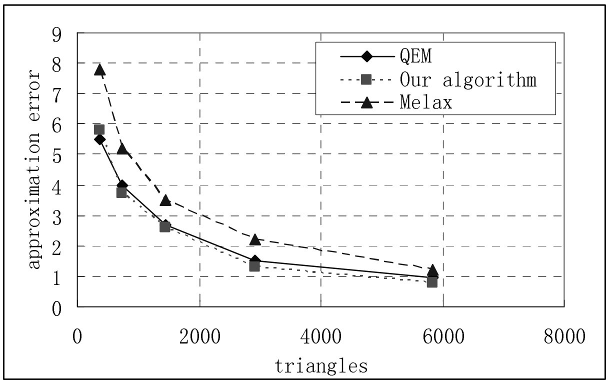


Fig. 6. Absolute maximum geometric error for lamp model. The size of bounding box is 15.6*15.6*22.6.

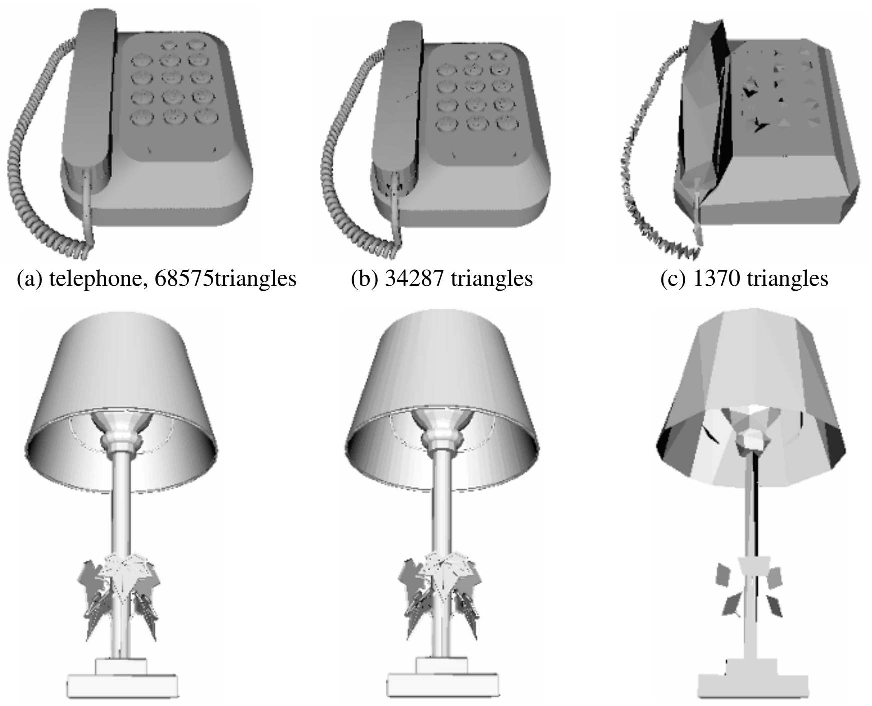
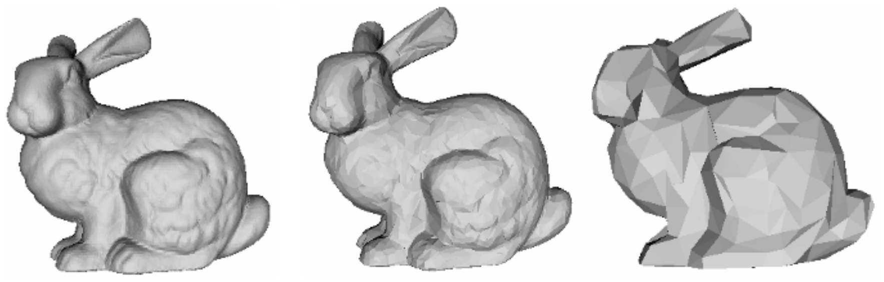


Fig. 7. The visual quality of the approximations generated using our algorithm. The bunny model (i) is drastically simplified (99%), but the major details still remain.



(g) bunny, 69451triangles (h) 6945 triangles (i) 694 triangles

Fig. 7. *(continued)*

6 Conclusion

We have presented a surface simplification algorithm which is capable of rapidly producing high fidelity approximations of 3d meshes. Our algorithm can preserve the visually important features of the model.

We also applied generic probabilistic optimization principle of Multiple-Choice algorithm to the problem of finding a simplification sequence. Experiments show that the MCA approach can reduce the memory overhead and lead to a simpler algorithmic structure.

Based on the proposed algorithm, we have implemented a simplification system. We have processed many 3D meshes of different sizes on this system, and achieved encouraging results. This demonstrates the effectiveness of our algorithm.

References

1. Campbell, F. W., Robson, J. G.: Application of Fourier Analysis to the Visibility of Gratings. Journal of Physiology 197 (1968) 551-566
2. Blakemore, C., Campbell, F. W.: On the Existence of Neurons in the Human Visual System Selectively Sensitive to the Orientation and Size of Retinal Images. Journal of Physiology, 203 (1969) 237-260
3. Wu, J., Kobbelt, L.: Fast Mesh Decimation by Multiple–choice Techniques. In Vision, Modeling and Visualization. IOS Press (2002) 241–248
4. Turk, G.: Re-tilling Polygonal Surfaces. In Proceeding of ACM SIGGRAPH (1992) 55-64
5. Schoroeder, W.J., Zarge, J.A., Lorensen, W. E.: Decimation of Triangle Meshes. In Proc. Of ACM SIGGRAPH (1992) 65-70
6. Rossignac, J., Borrel, P.: Multi-resolution 3D Approximation for Rendering Complex Scenes. In Geometric Modeling in Computer Graphics Springer Verlag (1993) 455-465
7. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. A., Stuetzle, W.: Mesh optimization. Computer Graphics (SIG-GRAPH '93 Proceedings) (1993) 19–26
8. Hoppe, H.: Progressive Meshes. In SIG-GRAPH 96 Conference Proceeding. ACM SIGGRAPH Addison Wesley August (1996) 99-108
9. Cohen, J., Varshney, A., Manocha, D., Turk, G.: Simplification Envelopes. In Proc. Of ACM SIGGRAPH '96 (1996) 119-128

10. Derose, T., Lounsbery, M., Warren, J.: Multiresolution Analysis for Surfaces of Arbitrary Topology Type. Technical Report TR 93-10-05 Department of Computer Science University of Washington (1993)
11. Eck, M., Derose, T., Duchamp, T., Hoppe, H., Lousbery, M., Stuetzle, W.: Multiresolution Analysis of Arbitrary Meshes. In Proceeding of ACM SIGGRAPH (1995) 173-182
12. Garland, M., Heckbert, P. S.: Surface Simplification Using Quadric Error Metric. In Proc. SIGGRAPH'97 (1997) 209-216
13. Agarwal, P., Suri, S.: Surface Approximation and Geometric Partitions. In Proceedings of 5th ACM-SIAM Symposium on Discrete Algorithms (1994) 24-33
14. Azar, Y., Broder, A., Karlin, A., Upfal, E.: Balanced Allocations. SIAM Journal on Computing, 29(1) (1999) 180-200
15. Kolchin, V., Sevastyanov, B., Chist-yakov, V.: Random Allocations. John Willey & Sons (1978)
16. Melax, S.: A Simple, Fast, and Effective Polygon Reduction Algorithm. Game Developer November (1998) 44-49
17. Southern, R., Blake, E., Marais, P.: Evaluation of Memoryless Simplification. Technical Report CS01-18-00, University of Cape Town (2001)