A General Simplification Algorithm

Boštjan Pivec, Vid Domiter

***Abstract*— In this article a new general algorithm for triangular mesh simplification is proposed. The algorithm extends Krivograd's work from 2D to 3D. For faster execution times a hash table is used. The main idea of the algorithm is based on vertex removal approach. With this approach we remove visually less important vertices. To determine their visual importance, all vertices have to be evaluated. This way models still preserve their essential characteristics. With simplification we can also easily present and transfer models over the network.

***Keywords*— average plane, hash table, simplification, triangular mesh.

I. Introduction

THE majority of free-form geometric objects are represented with triangular meshes. Due to high performance of today's computers, meshes can be described with more than 1.000.000 triangles. Such meshes can be rendered in real-time even on low-cost personal computers. However, the problem occurs when transferring such large meshes over the network. Namely, beside geometric data (coordinates of vertices), triangular meshes are also described by topological information, which defines how triangles fit together. Hence, it follows that special-purpose methods for topology compression have been proposed [1], [2]. Fortunately, high precision of transferred triangular meshes is not always required. Sometimes we are satisfied only with good visual representation of the geometric model. So we can afford to lose some data with simplification. There are three main approaches to choose from:

- The most frequently used methods are based on Schroeder's et. al. simplification algorithm [3], [4]. First, vertices are evaluated and then incrementaly removed from the mesh according to their importance.
- Edge simplification methods evaluate and remove edges. Removed edge is replaced with a vertex [5], [9].
- Simplification based on triangles is possible in theory yet practial solutions have not been reported.

Manuscript received May19, 07, revised Dec.20, 07 This work supported by the University of Maribor, Faculty of Electrical Engineering and Computer Science. The first author received research grant for one year.

B. Pivec and V. Domiter are with the Computer Science Department, Faculty of Electrical Engineering and Computer Science, University of Maribor, 2000 Maribor, Slovenia (e-mail: bostjan.pivec@uni-mb.si, vid.domiter@uni-mb.si, web: http://gemma.uni-mb.si).

The algorithm presented in this article is based upon Schroeder's vertex simplification method [3]. To speed up the search for the most suitable vertex to be removed, a hash table is used [6]. Krivograd et. al. suggested this approach for 2.5D triangular meshes [7]. The presented algorithm expands Krivograds's work in 3D.

The rest of the paper is organized as follows. In section 2 the algorithm in general is described. Most important steps of the algorithm are presented in the following subsections. In section 3 the results are discussed and section 4 concludes the paper.

II. The Algorithm

The algorithm supports 2D, 2.5D and 3D triangular meshes. The method, which our algorithm is based on, calculates weights for all vertices, before they can be removed. The input of the algorithm are a list of vertices and a list of triangles. The algorithm works in the following steps:

1. Evaluation of all vertices (computing weights).
2. Arrangement of all vertices into a hash table.
3. Selection of the most proper vertex for removal.
4. Removal of the selected vertex from the hash table and triangular mesh.
5. Removal of the surrounding triangles of the removed vertex.
6. Triangulation of an empty space; empty space is a consequence of removed triangles.
7. Reevaluation of neighbouring vertices of the removed vertex.
8. Rearrangement of reevaluated vertices in the hash table. Returning to step 3 until the final condition is reached.

The final condition depends on user's decision about visual quality of the simplified model. User can stop the simplification process at a desired level of detail. The algorithm also stops when only the last list of vertices remains in the hash table (see section 2.3).

A. *Evaluation of Vertices*

The first step of our algorithm is the evaluation of all vertices in a triangular mesh. This way each vertex is evaluated according to its visual importance. Smaller is the evaluation factor, the less important is the vertex. There are different possibilities for the evaluation criteria. For example, in 2.5D, two approaches are proposed [7]:

- The vectors connecting a chosen vertex and its neighbouring vertices are constructed. After that, angles between these vectors and the appropriate plane is calculated. For 2.5D triangular meshes this plane is XY plane. The average value of all angles between this plane and vectors, which are defined by chosen vertex and its neighbouring vertices, is considered as an evaluation factor.

- All distances from evaluated vertex and its neighbouring vertices to XY plane are computed. The evaluation factor is an average difference of all distances between chosen vertex and its neighbouring vertices regarding the XY plane.

In our approach, the second possibility is generalized to 3D triangular meshes. Instead of using the XY plane, we use an average plane on which the evaluation factor is calculated.

The evaluation factor can represent the distance between the evaluated vertex and the line defined by its neighbouring vertices, if the evaluated vertex has only two neighbours. This case can occur only in the corners of non-closed triangular meshes. The most frequent situation appears when the evaluated vertex has three or more neighbouring vertices. In this case, the average plane has to be calculated.

The first step in determination of the average plane is the calculation of all possible combinations of three arbitrary neighbouring vertices. Each combination defines a plane in the space described by a normal vector. The average normal vector, which determines the average plane, is calculated as the avarage value of all normal vectors. Thus, the evaluation factor represents the average distances between the selected vertex (Fig. 1) and its neighbouring vertices regarding the average plane.

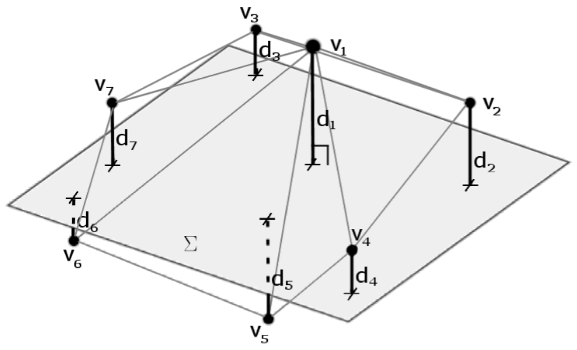

Figure 1: evaluation of vertex v_1 according to average plane Σ

B. Visual Importance

One of the aims of the simplification algorithm is to keep essential characteristics of the simplified model. With simplification we lose some data from our triangular mesh. It is very important that the data we lost during the process do not affect the main characteristics too much. The data which do not noticeably affect main characteristics are visually important. To keep visually important data we associate each vertex with a value that represents its visual importance. This value is called evaluation factor (see section 2.1).

In Fig. 2 a simple example of visual important vertices in 2.5D is shown. Vertices that have higher evaluation factor also have a higher visual importance (vertex v_{13} in Fig. 2). These vertices are usually farther away from the average plane or plane XY as is the case in the Fig. 2. Vertices that are closer to the plane would cause the smallest visual change on the model so they can be removed (vertices from v_9 to v_{12}). In case the model is not manifold it is important to preserve vertices that are connected to its boundary edges (vertices from v1 to v8).

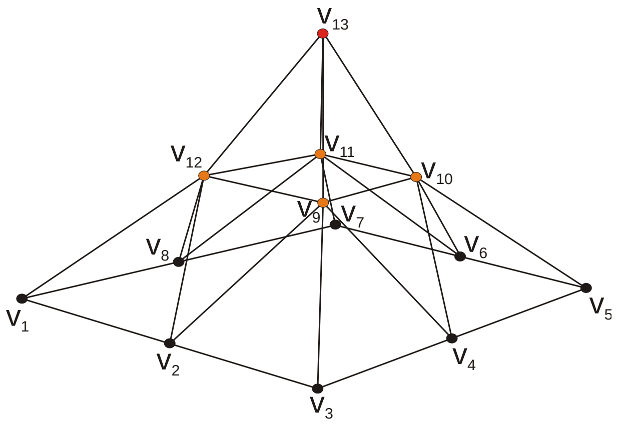

Figure 2: Example of vertex visual importance

C. Hash Table

The smallest visual change in triangular mesh is caused by deleting a vertex with the smallest evaluation factor. After deleting such a vertex, the evaluation factors of its neighbouring vertices has to be recomputed. After that, the vertex with the smallest evaluation factor has to be found again. In this way, $O(n^2)$ time complexity is reached. To speed up the selection of the vertex for its removal, a hash table is applied as suggested by Franc and Skala [6] (Fig. 3). Vertices are inserted in the hash table according to their evaluation factors. The same heuristics for constructing the hash table as proposed in [7], has been used.

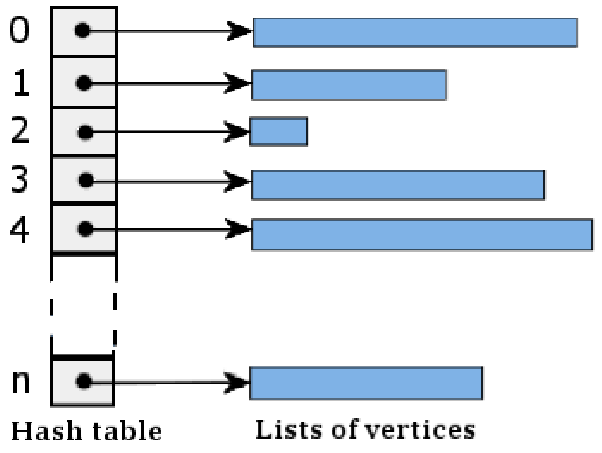

Figure 3: Structure of the hash table where **n** is the number of intervals

The algorithm always selects the first vertex from the lowest non empty hash table entry. The surrounding neighbours of removed vertex have to be evaluated again. Normally, reevaluation change their evaluation factor and sometimes also the interval in the hash table. The reevaluated vertices are always placed at the end of the list of vertices in the corresponding interval of the hash table. This way, the local simplification of the triangular mesh is prevented.

The hash table is also used to preserve the basic shape of geometric objects. This can be accomplished by preserving the last list of vertices in the hash table, where the edge vertices and vertices with the highest evaluation factors are stored.

D. *Triangulation*

All triangles defined by deleted vertex are removed too. The result is a hole in the triangular mesh which needs to be filled with new triangles. The outer edges of the removed triangles form a polygon which has to be triangulated. This polygon is not necessarily planar. Therefore, all vertices of this polygon are mapped to the average plane (see section 2.1). In Fig. 4 vertices from v_2 to v_6 are projected onto vertices v'_2 to v'_6 of the planar polygon.

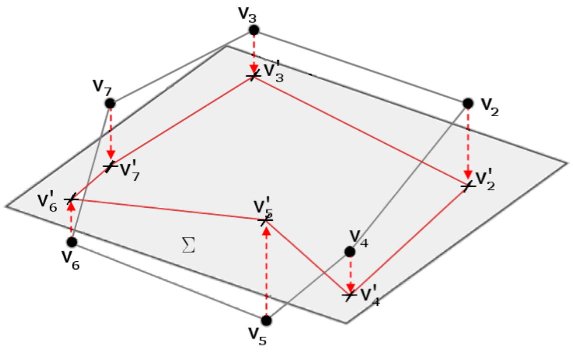

Figure 4: Projection of vertices on average plane

If the polygon is convex, the triangulation is trivial, but if it is concave, the well known ear cutting method [8] is applied. In Fig. 5 we can see a triangulation of the concave polygon.

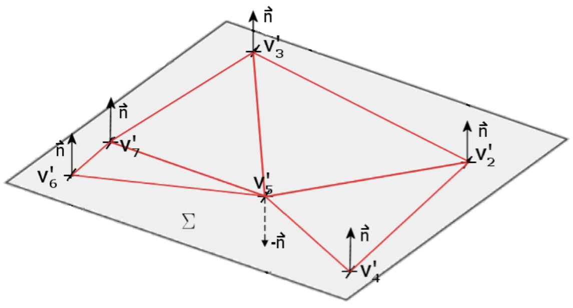

Figure 5: Triangulation of concave polygon

When the triangulation of the planar polygon is finished the newly constructed triangles are mapped to the triangular mesh. Fig. 6 shows the innitial polygon filled with new triangles.

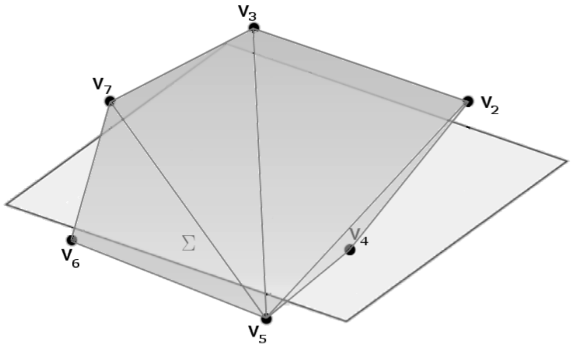

Figure 6: Result of the triangulation

III. Results

In this chapter we estimate theoretical time complexity and show practial results of the algorithm. The time complexity is as follows:

- Evaluation of vertices. *l* neighbouring vertices of the evaluated vertex are needed for calculation of the evaluation factor. As $l<<n$ we can suppose that constant time *O(1)* is needed. Therefore all vertices are evaluated in time *O(n)*.
- Hash table. Constructing the hash table and filling it with vertices is done in linear time *O(n)*. Each step removes selected vertex from the hash table in constant time *O(1)*. There are $l << n$ reevaluated vertices. Each is inserted into the hash table in constant time as well.
- The triangulation using the ear cutting method of a polygon having *l* vertices is built in time *O(l)*. As $l << n$ the task can be considered as done in constant time *O(1)* per removed vertex.

Therefore the common time complexity of the algorithm is $O(n)$.

The algorithm was tested on various triangular meshes. We simplified meshes to a level that still assures good visual quality. Results of the simplification of Pumpkin and Horse models are shown in Table I. The time graphs in Fig. 7 confirms good behaviour of the algorithm.

The simplification of the Pumpkin model is presented in figures 8, 9 and 10. In figures 11, 12 and 13 the simplification of the Horse model is shown.

All measurements were performed on a computer with AMD Athlon XP 2800+ processor and 1GB DDR 333Mhz RAM.

TABLE I:
SIMPLIFICATION RESULTS FOR PUMPKIN AND HORSE MODEL

Mesh	%	Vertices	Triangles	Time(s)
Pumpkin Vertices: 5002 Triangles: 10000	90	4502	9000	0.062
	80	4002	8000	0.125
	70	3502	7000	0.188
	60	3002	6000	0.25
	50	2501	4998	0.297
	40	2001	3998	0.359
	30	1501	2998	0.406
	20	1001	1998	0.438
	10	501	998	0.484
Horse Vertices: 48485 Triangles: 96966	90	43647	87270	6.469
	80	38788	77572	13.969
	70	33940	67876	20.782
	60	29091	58178	26.312
	50	24243	48482	29.516
	40	19394	38784	32.359
	30	15242	30480	33.437
	20	9725	19448	34.61
	10	4876	9754	35.265

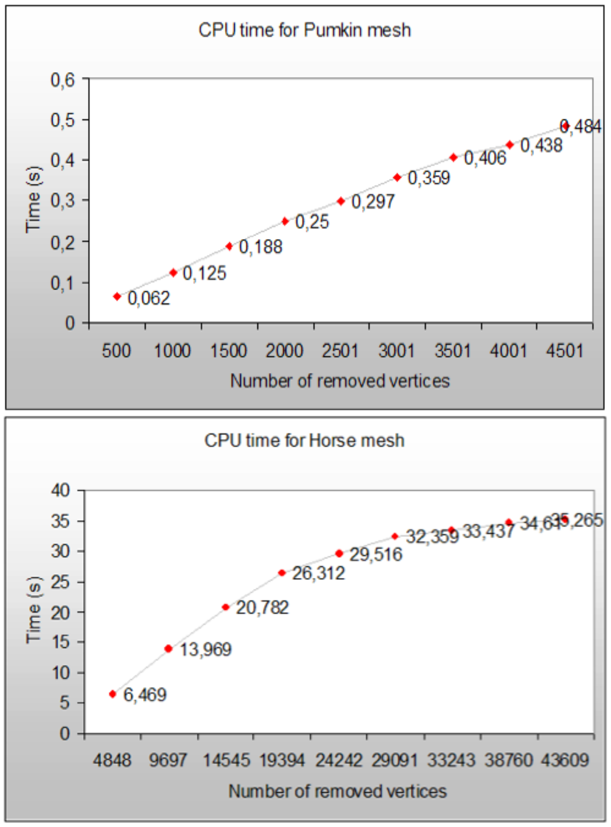

Figure 7: Time graphs for simplification of the Pumpkin and Horse models

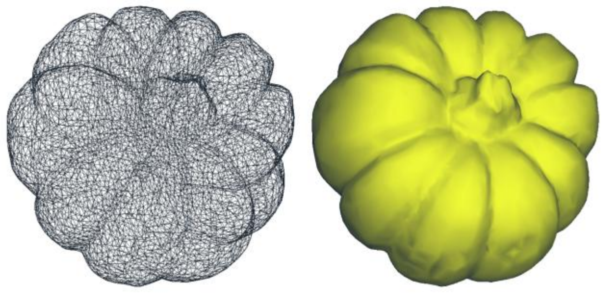

Figure 8: Pumpkin model with 5002 vertices and 10000 triangles

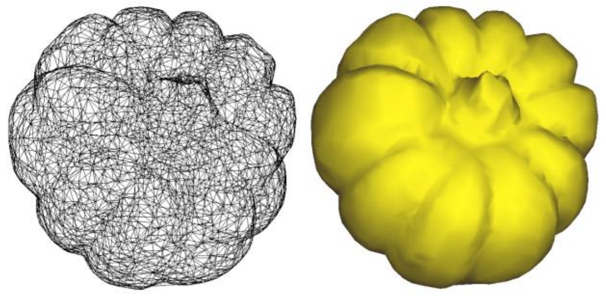

Figure 9: Simplified Pumpkin model at 50%

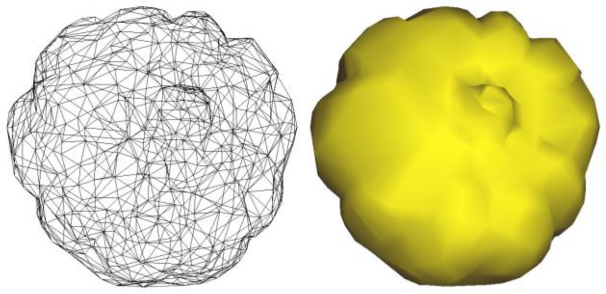

Figure 10: Simplified Pumpkin model at 10%

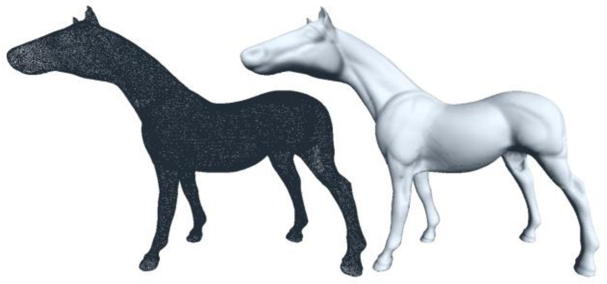

Figure 11: Horse model with 48485 vertices and 96966 triangles

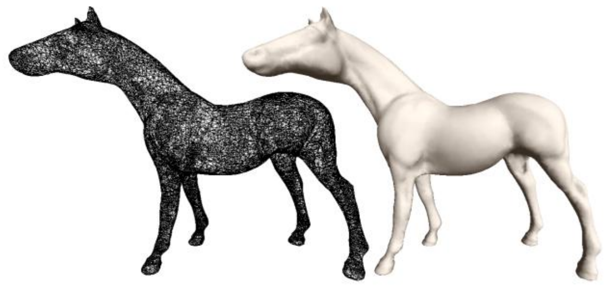
Figure 12: Simplified Horse model at 50%

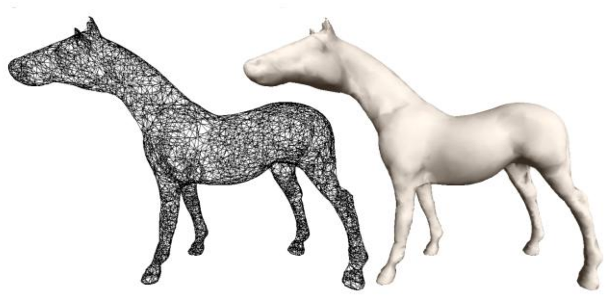
Figure 13: Simplified Horse model at 30%

IV. Conclusion

In this article a simple and efficient algorithm for model simplification is presented. The algorithm is based on the vertex removal method. It is common for this method that all vertices have to be evaluated before simplification. Afterwards, vertices are arranged into a hash table according to their evaluation factor. This factor represents visual importance of the vertex. Less visually important vertices have lower evaluation factor and are placed at the beginning of the hash table. When all vertices are arranged in a hash table, the simplification process can begin. Process can be stopped by the user at any level of detail, or it stops automatically when all vertices have been removed except those in the last interval of the hash table. The hash table significantly speeds up the simplification process.

The algorithm supports 2D, 2.5D and 3D triangular meshes that are used to represent geometrical models. With use of simplification we can lower the level of detail of models so they can be easily transferred over the network.

References

[1] C. Touma, C. Gotsman, "Triangle Mesh Compression", *Graphics Interface*, 1998, pp. 26-34.

[2] P. Alliez, M. Desbrun, "Valence-Driven Connectivity Encoding for 3D Meshes", *Computer Graphics Forum*, Vol. 20, No. 3, 2001, pp. 480-489.

[3] William J. Schroeder, Jonathan A. Zarge, William E. Lorensen, "Decimation of Triangle Meshes", *Computer Graphics*, Vol. 26, No. 2, 1992, pp. 65-70.

[4] Michael Garland, Paul S. Heckbert, "Fast Polygonal Approximation of Terrains and Height Fields", *technical report*, CMU-CS-95-181, 19. September 1995.

[5] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, Werner Stuetzle, "Mesh Optimization", *ACM Computer Graphics – SIGGRAPH'93*, Anaheim, California, Unated States, 1993, pp. 19-26.

[6] M. Franc, V. Skala, "Parallel Triangular Mesh Decimation Without Sorting", *SCCG Proceedings*, Budmerice, 2001, pp. 69-75.

[7] Sebastian Krivograd, Borut Žalik, Franc Novak, "Triangular mesh decimation and undecimation for engineering data modeling", *Inf. MIDEM*, Vol. 32, No. 3, September 2002.

[8] Marko Lamot, Borut Žalik, "A fast polygon triangulation algorithm based on uniform plane subdivision", *Computer & Graphics*, Vol. 23, No. 2, 2003, pp. 239-253.

[9] Muhhamad Hussain, Yoshihiro Okada, Koichi Niijima, "Efficient and Feature-Preserving Triangular Mesh Decimation", *WSCG 2004*, 2004, pp. 167-174.