Proceedings Spring Conference on Computer Graphics SCCG2006. Bratislava : Univerzita Komenského, 2006. s. 136-143. ISBN 80-223-2175-3.

Mesh simplification with respect to a model appearance

Martin Franc† Václav Skala‡
{marty,skala}@kiv.zcu.cz

Department of Computer Science
University of West Bohemia, Plzeň, Czech Republic

1 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 computations we have to handle models having hundreds thousands or even millions of triangles. The source of such models are usually 3D scanners, computer vision and medical visualization 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 modeled 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 an original algorithm for mesh simplification with respect to similarity of appereance of the original model and resulting approximation. In this approach we search for important features of the model which must be preserved during simplification process. Since vertices defines these features, we combine techniques of vertex and edge decimation. We introduce original method for new vertex position estimation as a part of edge collapse.

This paper is structured as follows: In section 2 we define a polygonal model, introduce error metric E_{avg} and discuss our previous work in the context of vertex decimation. In section 3 we present four proposed methods how to evaluate vertex properties defining model features. Section 4 discusses reached results and shows some figures. At last, section 5 provides a conclusion with respect to future work.

2 Motivation

As already mentioned, 3D models are often represented by a polygonal surface. We use Garland's definition [3]. A polygonal model M is composed of a fixed set of vertices $V = (v_1, v_2, \ldots, v_k)$ and a fixed set of faces $F = (f_1, f_2, \ldots, f_n)$. It provides a single fixed resolution representation of an object. Without loss of generality, we can assume that the model consists entirely of triangular faces, since any non-triangular polygons may be triangulated in a pre-processing phase. To streamline the discussion, we will assume that the models do not contain isolated vertices and edges.

The goal of simplification is to get a surface approximation M_2, which is as close as possible to the original surface M_1. Therefore we need some means of qualifying the notion of similarity. The usual way is to evaluate the approximation error $E(M_1,M_2)$. Probably the most popular approach is to compute a geometric error using E_{avg} metric (1,2) derived from Hausdorff distance:

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

$$d_v(M) = \min_{w \in P(M)} \|v - w\|,$$

where M_1 and M_2 are original and reduced model and k_1 and k_2 are numbers of vertices on each model.

If $E_{avg}(M_1,M_2) < \varepsilon$ then we know that every point of the approximation is within ε of the original surface and that every point of the original is within ε of the approximation.

Since vertex decimation methods [7] produce good quality results and preserve a mesh topology, we used this approach in our previous work too. In vertex decimation algorithm all vertices V in a mesh M are at first classified

† **Supported by the project 6FP EU NoE 3DTV No.511568**
‡ **Supported by the project MSMT CR, Project LC06008**

according to their topology into five sets. Simple, border, complex, corner and vertices on sharp edges. Except of complex vertices their importance is evaluated. The common technique is to compute the distance of the vertex from the average plane given by its neighborhood (see Fig. 2). Vertices are then sorted according to their importance and less evaluated vertices are removed. The resulting hole is re-triangulized afterwards.

Given the main task to find a fast and robust solution, we used a combination of vertex decimation [1] and a kind of half-edge collapse method [2] to make the re-triangulation easier and more safety (we can easily detect and avoid when triangle folds over itself). Thanks to parallel implementation and some improvements we reached quite good times, and good quality of the resulting approximation, by mean of geometric error E_{avg}.

However, there are many problems connected with vertex decimation in general. The first thing is that it is necessary to explicitly define the sharp edges. Sharp edge is the edge, where the angle between the two adjacent triangles is less than the specific threshold. To set the threshold some experience of the user is necessary and such a threshold is different for each model. The main disadvantage of vertex decimation methods is that they are not able to preserve a volume, since they produce shrinkage of the reduced model (assuming closed surfaces with a majority of convex vertices). Figure 1 illustrates the situation in 2D. The more the model is reduced, then the smaller is its volume, or area respectively. Unfortunately there is no way how to avoid this effect. Therefore methods based on edge collapse seem to be more promising from this point of view.

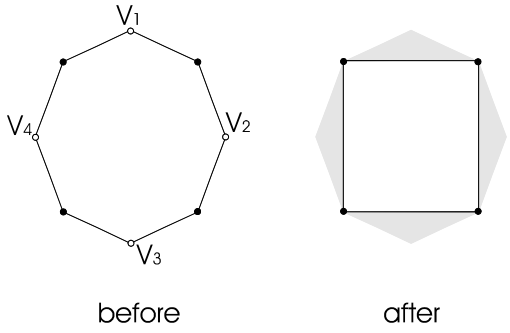


Fig. 1: A shrinkage caused by the removal of vertices v_1, v_2, v_3 and v_4.

Moreover, edge contraction methods offer intuitive techniques for eliminating approximation error by optimal positioning of a new vertex after performing an edge collapse. These algorithms [10,8,5,6] have become very popular in recent years. In [4] the edges are generalized to vertex pairs. Each pair is evaluated according to a quadric error caused by further collapse of this pair into one vertex. Such edges are put into a priority queue and iteratively collapsed until required amount of simplification or given error value is reached.

Our new approach looks at the simplification problem from a slightly different point of view. Although a geometrical distance can tell a lot about model similarity in space, for a similar visual appearance we would rather follow some other criteria. In general, the proosed approach is based on detection of the main features of original model and applying few heuristic rules tries to keep these features over whole simplification process.

3 Vertex estimation – feature detection

As a feature we consider either an extreme vertex (a peak) or a sharp edge (two or more extreme vertices) – in other words, what a human eye is sensitive to. Thus features detection is naturally based on vertex evaluation. We do not study any properties of edges or triangles as they are defined in the model. We suppose that these elements (edges and triangles) are derived from original set of points anyway. Although we detect feature vertices, we of course mainly search for non-important vertices – vertices to be easily removed with a minimal harm on model's appearance. The best candidates for removal are the least important vertices being part of planar regions of the mesh.

We have studied 4 approaches how to evaluate vertex importance.

3.1 Average plane distance

First method was based on evaluating vertex property according to previously used distance from an average plane. The plane is given by vertices adjacent to evaluated vertex $\boldsymbol{v}$. This is the same technique which is used for vertex importance computation in vertex decimation methods [1,7], see Fig. 2.

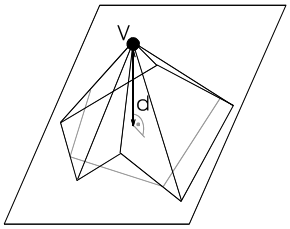


Fig. 2: Distance to the average plane

An average plane is constructed using the triangle normals $\mathbf{n}_i$, midpoints $\mathbf{x}_i$ and areas A_i.

$$\vec{N} = \frac{\sum_{i=1}^{m} \vec{n}_i A_i}{\sum_{i=1}^{m} A_i}, \quad \vec{n} = \frac{\vec{N}}{\left|\vec{N}\right|}, \quad \vec{x} = \frac{\sum_{i=1}^{m} x_i A_i}{\sum_{i=1}^{m} Ai} \qquad (3,4,5)$$

where the summation is over all triangles in the vertex neighborhood.

The distance *d* of the vertex ***v*** from the plane is then

$$d = \left|\vec{n} \bullet \vec{v} - \vec{x}\right| \qquad (6)$$

and its value is taken as a vertex property. The higher the distance is the more important is the vertex in a model. Vertices with high values are good candidates to be marked as feature vertices. Vertices with near-to-zero distance can be removed

3.2 Gaussian curvature

Since we mostly search for planar regions, we also did several experiments using Gaussian and mean curvature estimation of the surface [11]. Because of our focus on flat areas and search for vertex pair with the same evaluation, the Gaussian curvature only was sufficient.

$$K = \frac{2\pi - \sum_{i=1}^{m} \alpha_i}{\frac{1}{3}\sum_{i=1}^{m} A_i} \qquad (7)$$

The curvature *K* is given by the equation 7, where *i* goes over all neighbors of evaluated vertex, α is the vertex angle in each of neighboring triangle and *A* means the area of neighboring triangles.
Note that we are searching for single vertex property only, thus we don't need to classify the mesh geometry exactly.

3.3 Volume estimation

Another widely used criterion in mesh simplification [8] is based on underlying condition to keep the volume of the original model. In this approach a vertex importance is related to the volume of the mesh below the vertex (part of the mesh given by adjacent triangles). For each vertex and its neighbourhood we introduce a new vertex v_v, given as an average point of all vertices adjacent to the vertex in question v_0.

$$\vec{v}_v = \frac{\sum_{i=1}^{m} \vec{v}_i}{m} \qquad (8)$$

Having this "virtual" vertex v_v we compute the a of volumes of tetrahedral v_0,v_v,v_i,v_j, where v_v is the virtual vertex and v_0,v_i,v_j are vertices of triangles in our triangulation. See Fig. 3.

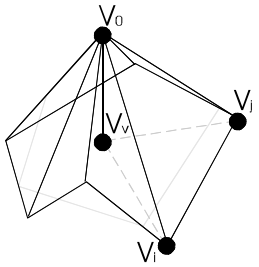


Fig. 3: Vertex related volume estimation

The resulting value is weighted by the longest edge going out from vertex v_0 to somehow normalize the values over the whole mesh. The volume has been evaluated according to following formula [9].

$$D_k = \begin{vmatrix} x_v & y_v & z_v \\ x_i & y_i & z_i \\ x_j & y_j & z_j \end{vmatrix}, \; V = \frac{1}{6}\sum_{k=1}^{m} |D_k|, \qquad (9,10)$$

where D_k is the volume of one tetrahedron and k goes over all tetrahedrons related to vertex v_0 which is supposed to lay in the origin.

3.4 Average normal vector

The last and in some way straight forward method evaluates vertices by estimating a normal vector in vertex v_0. The normal vector n is computed as an average normal of all triangles adjacent to the vertex v_0, see Fig. 4.

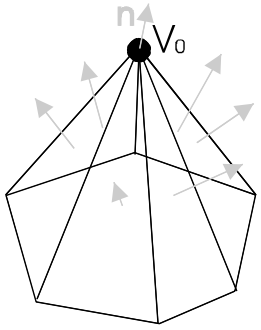


Fig. 4: Average normal vector

We used the easiest way of computation, which is not-weighted average, see equation (11).

$$\vec{N} = \frac{\sum_{i=1}^{m} \vec{n}_i}{m}. \qquad (11)$$

Note that the normals of triangles have unit length. The importance value is the inverse of the length of the normal. The more the normal length is closer to 1 the more flat area is around the vertex in question. Naturally, if all the neighbouring triangles have their normals in the same direction, the area of the triangle fan is flat and the length of resulting normal will be equal to 1. The more the vertex represents a peak in a mesh the less will be the resulting normal length, since each of partial normals point to different direction.

4 Final algorithm

Evaluation results

Studying the results of presented evaluation, we have decided to use the average normal for vertex importance estimation (feature detection). On figure 5 you can see the example of cow model with 50% most important vertices highlighted according to all methods presented. Since all the pictures show exactly 50% most important vertices, it is obvious, that the average normal estimation (top left) gives the best results showing the main features of the model. As you can see, it is a kind of caricature, where the most important contours are highlighted (horns, ears, eyes, neck, and legs). The Gaussian curvature estimation (top right) also gives good results which could be even better with combination of mean curvature to detect sharp edges too instead of peak points only. However, the computation would be too time-consuming. Average distance evaluation (bottom left) tends to involve the sharp edges too. On the other hand it misses the details kept by small triangles in areas such as eyes and also highlights which could be supposed not to be very important such as a belly. Probably the worst result gives the estimation of tetrahedrons volume, which was anyway more or less experimental.

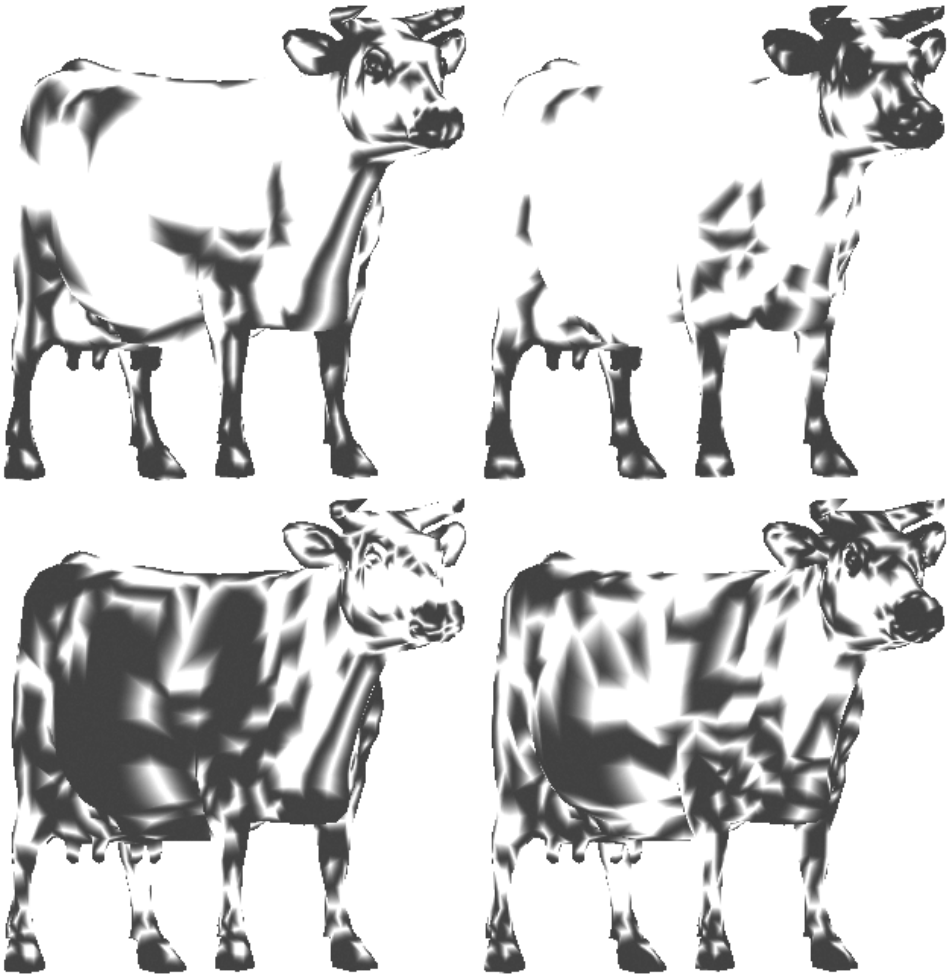


Fig. 5: 50% important vertices of cow model according to 4 different evaluation used – average normal (top left), Gaussian curvature (top right), average plane distance (bottom left), tetrahedral volume (bottom right).

The graph on Fig. 6 shows the rate of vertices and their importance. The picture says that from all the number of vertices (approx. 3000) there are about 2000 vertices with importance lower then 0.5. It's obvious, that all the importance evaluation methods act the similar way and the majority of vertices has a low importance (tests has been performed on several models naturally). Again, the average normal vector evaluation gives the most wanted results - declaring the majority of vertices as non-important (the lowest line).

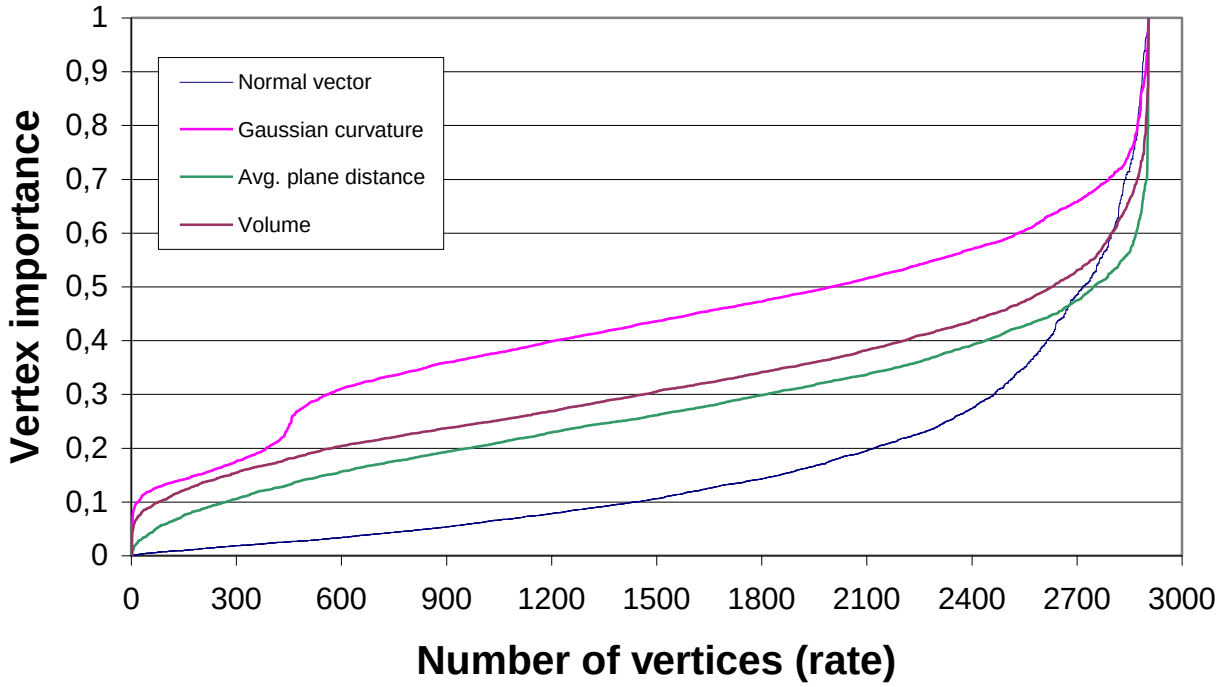


Fig. 6: Vertex evaluation according to the method used in a model of a cow.

After several experiments with the average normal computation (not weighted, weighted by triangle area), we concluded to the evaluation, where each normal is weighted by the apical angle of given triangle. The precise formula can be seen on equation (12).

$$\vec{N} = \frac{\sum_{i=1}^{m} \alpha_i \vec{n}_i}{\sum_{i=1}^{m} \alpha_i}, \qquad (12)$$

where m is the number of neighboring triangles and α_i is the apical angle of i-th triangle at the vertex (see Fig. 7). Note that the normal vector is not normalized.

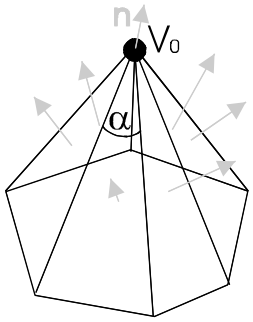


Fig. 7: Normal vector weighted by apical angle α for each of adjacent triangle.

This approach produces best quality evaluation which is independent on tessellation at the vertex (see Figure 8). In case of non-weighted normal, the resulting normal vector will be different in example on the left (the other two will be the same and the normal will have a direction more to the front). If the normal vector would be weighted by triangle area, the first two examples will have the same normal, but the third one will be different (since the area is smaller). Only the apical angle weight will give us the same results for all three cases.

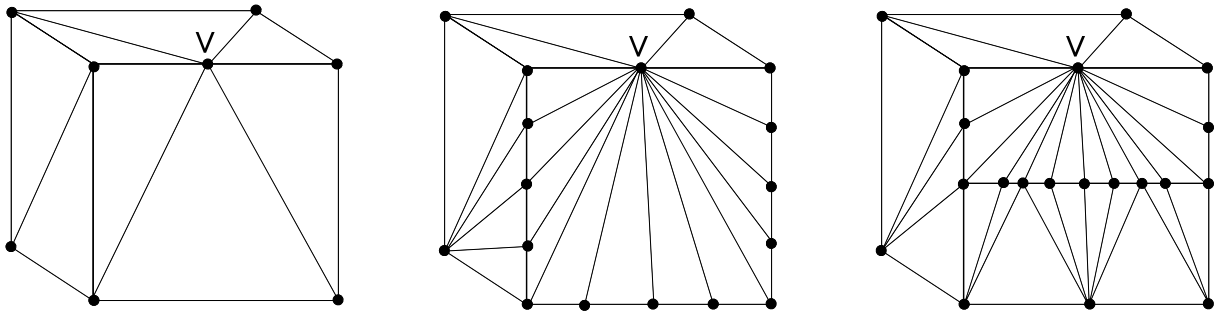


Fig. 8: Three examples of vertex neighbourhood.

Having evaluated all the vertices we can sort them in priority queue according to their importance. The least important vertices are the best candidates for removal. Since vertex removal following by re-triangulation is neither trivial in 3D nor natural when ignoring original edges, we perform and edge contraction instead. For a given vertex the adjacent edges are investigated and the best-fitting one is replaced by a new vertex. By this step we get a correct triangulation and are able to follow some other criteria on a quality of resulting mesh.

Best-fitting edge estimation

As already said, once we have a vertex candidate for the removal, we need to search for the best edge to perform the contraction. The chosen edge will be replaced by a new vertex of specific position. Before we describe the edge estimation, we must first introduce an evaluation of new vertex position.

Since the aim of the method is to find a simplified approximation of the model with respect to the similarity of appearance, we don't subordinate the vertex position to any error estimation. We try to find such a vertex which would approximate the suppositional surface in between the two end-points of the edge, seen Fig. 9.

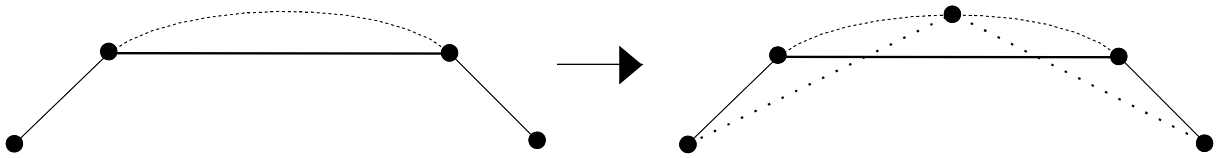


Fig. 9: The new vertex should lie somewhere on the dashed line (in 2D).

Let's consider 2D case for clearer explanation. To determine a new vertex position we use a curve which approximates surface the way we show on the picture 9. At the beginning we have only two endpoints and non-normalized normals which determine vertices importance. We used a quadric curve with near least square acceleration, introduced originally to smooth the model contour by [12]. The nice thing about this curve is its invariance to tangent lengths. The tangents corresponding to a pair of normals at the vertices can be obtained by using the so-called Gram-Schmidt orthogonalization algorithm.

$$\mathbf{T}_1 = \mathbf{N}_2 - \mathbf{N}_1 (\mathbf{N}_1 \cdot \mathbf{N}_2), \qquad (13)$$

$$\mathbf{T}_2 = -\mathbf{N}_1 + \mathbf{N}_2 (\mathbf{N}_1 \cdot \mathbf{N}_2). \qquad (14)$$

Note, that the normals are assumed to be normalized. It should also be pointed out that when the angle between the normals, is zero or very close to zero, then we can not compute the tangent in this way. We use a linear interpolation on the edge instead. Let's have

$$\mathbf{P} = \mathbf{P}_2 - \mathbf{P}_1 \qquad (15)$$

and coefficients $\alpha, \beta, \alpha', \beta'$, where

$$\alpha = \frac{T_1 T_2}{T_1 T_1}, \beta = \frac{P\,T_1}{T_1 T_2}$$
$$\alpha' = \frac{T_1 T_2}{T_2 T_2}, \beta' = \frac{P\,T_2}{T_1 T_2} \qquad (16,17,18,19)$$

we get a function

$$f(t) = \left(\frac{\alpha'\beta' T_2 - \alpha\beta T_1}{2} \right) t^2 + \left(P + \frac{\alpha\beta T_1 - \alpha'\beta' T_2}{2} \right) t + P_1 \qquad (20)$$

Thus the only question is how to choose the parameter *t*, which means, to where to put a new vertex on the curve. Since the curve itself is given by normals as well as the vertex importance, the natural way is to do linear interpolation on the importance (length of the normal vectors). Thus, if the importance of vertices will be equal, the parameter *t* will be set to 0.5 (the new vertex will be placed in the middle of the curve).
Switching to 2.5D, we have a tool now how to place a new vertex somewhere "above" the contracted edge, instead of just somewhere in between the endpoints of the edge. In full 3D space we can also influence the vertex position by the properties of triangles adjacent to the edge, or their opposite corners respectively. We can use the quadric curve again and the only condition is that the area is not arrow-shaped, see Fig. 10. In case of arrow-shaped quads we use only one curve constructed over the edge.

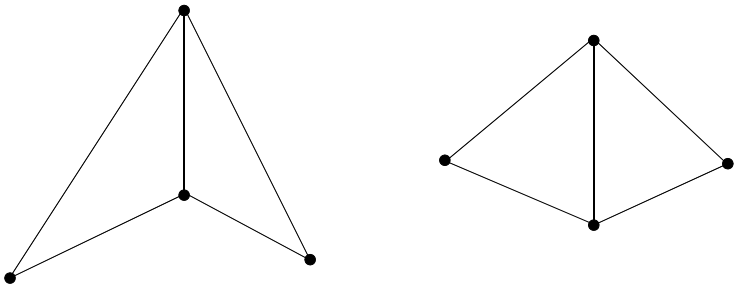


Fig. 10: The arrow-shaped quad (left), the non-problematic case (right).

The parameter *t'* of the second curve (between opposite corners of two triangles adjacent to the edge) is again given by the importance of opposite corner vertices, but this time it is weighted by the inverse value of the distance of each vertex to the point estimated on the first curve. So, if the distance is equal, then only the importance matters. If the distance of one vertex is higher the less of its importance is considered. This condition forces to place the vertex in the position given by the importance of surrounding vertices and not the exact topology of the mesh. The final vertex *rv* is placed just in the middle of the two vertices on both curves, see Fig. 11.

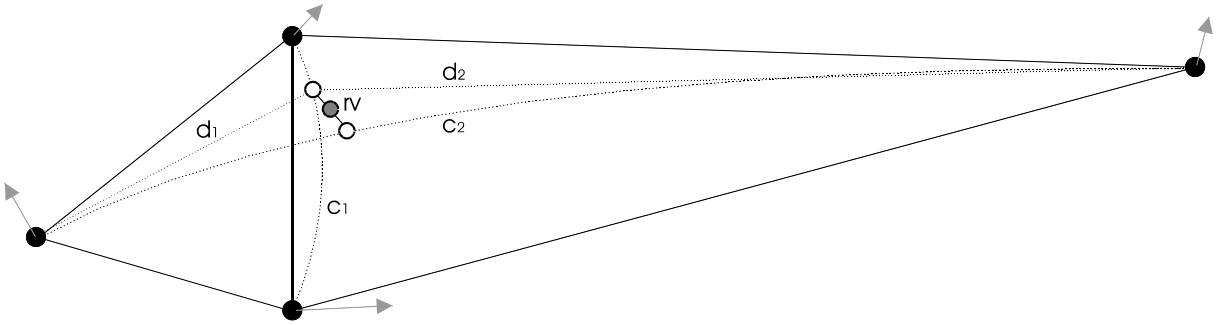


Fig. 11: The construction of new vertex position.

Having described the new vertex position evaluation, we can get back to the procedure of choosing the best-fitting edge for the contraction. In our algorithm we take all the edges adjacent to given vertex (candidate for removal), and compute new vertex position for each edge. For every new vertex we examine affected mesh property like the difference between area of original and resulting triangles, or inconsistencies such as mesh folding or triangle degeneration caused. Since we primarily remove vertices on flat regions, we force the resulting mesh to be as flat as possible, thus minimal area of resulting triangles is prioritized. The edge with such a best evaluation is contracted for real. If there is no suitable edge to process, the removal is forbidden.

Upon a framework described above, we can present the proposed algorithm as follows
Init:

- go over all the vertices and compute their importance
- sort vertices according to the importance

Main loop:

- take the least important vertex
- for every adjacent edge
  - compute new vertex position for case of contraction
  - simulate the contraction and evaluate the quality of resulting mesh
- perform the contraction of best-fitting edge
- re-evaluate affected area
  - compute vertex importance
  - insert into a priority queue
- continue the main loop

The real implementation uses three "magic" parameters which are important for the resulting mesh and can be changed by user. The first parameter is an importance threshold. If vertex importance exceeds the value of the threshold, it is marked as extremely important one. These (feature) vertices are processed a bit different way than described above. The main difference is in estimation of parameter *t*, which is set either to 0 or 1 depending on which endpoint of an edge is extreme. If both of them are extreme, the more important vertex wins and its position is kept. This strategy leads to simplification which keeps the most important features represented by extreme vertices without any change from the original mesh. Such extreme vertices can be seen on Fig. 12 for example at the neck. The lower is the threshold value the more features of the model will be kept and less simplification will be performed.

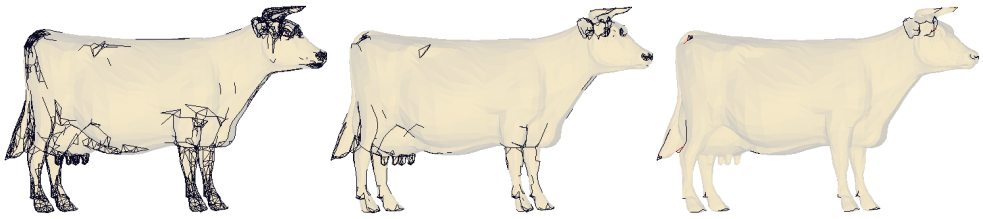


Figure 12: Model feature detection for (a) 50%, (b) 75% and (c) 90% simplification.

The second parameter is the maximal allowed angle between normals of triangles before and after edge contraction. This value helps to detect triangle folding and also controls the smoothness of the resulting mesh. The smaller is the angle the smoother is the resulting mesh – contractions producing not-wavy surfaces are preferred.
Third parameter is an angle between two adjacent triangles and helps to define so called flat edge. In general if the vertex selected for removal is extreme and one of its neighbourhood vertices is extreme as well, only the edge between these two vertices can be considered for removal. Applying this rule we can preserve sharp[§] edges. In this case a sharp edge is every edge that has extreme endpoints and is not a flat edge. In other words, we do not detect sharp edges studying the sharp angle between adjacent triangles. The algorithm marks the edge as sharp if both of its endpoints are

[§] An edge, where the angle between its two adjacent triangles is lower than some specific value.

extreme and the angle between adjacent triangles is less than the value given by our third magic parameter. If such angle is bigger (at most 180 degrees) the edge is marked as a flat edge and is prohibited from contraction, since it could dramatically change the shape represented by the mesh, see Fig. 13.

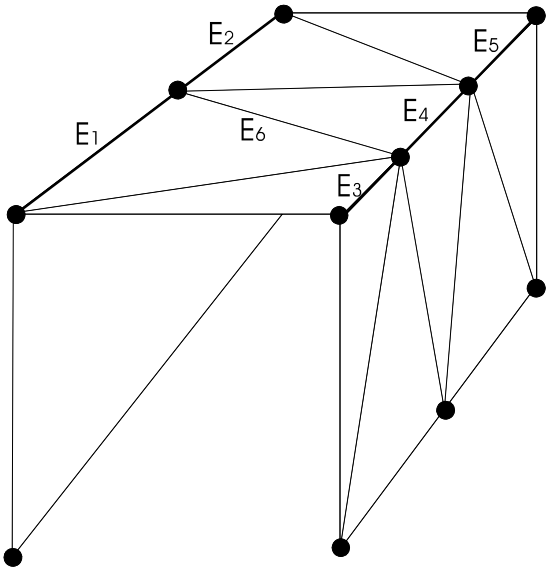


Fig. 13: An example of sharp edge (E_1-E_5) and flat edge (E_6).

In general two strategies can be for simplification process - with or without memory of reduced vertex and affected area. The approach with memory initializes a counter of affected vertex during reduction and every time the vertex is marked as a candidate for removal the counter is decreased. Only vertices with a counter equal to zero can be considered for reduction. This memory helps to distribute reduction over whole mesh and the resulting mesh has nicely shaped triangles. If the simplification runs without memory it can easily produce rapid-flat models, where flat regions are simplified in prior, see Fig. 14.

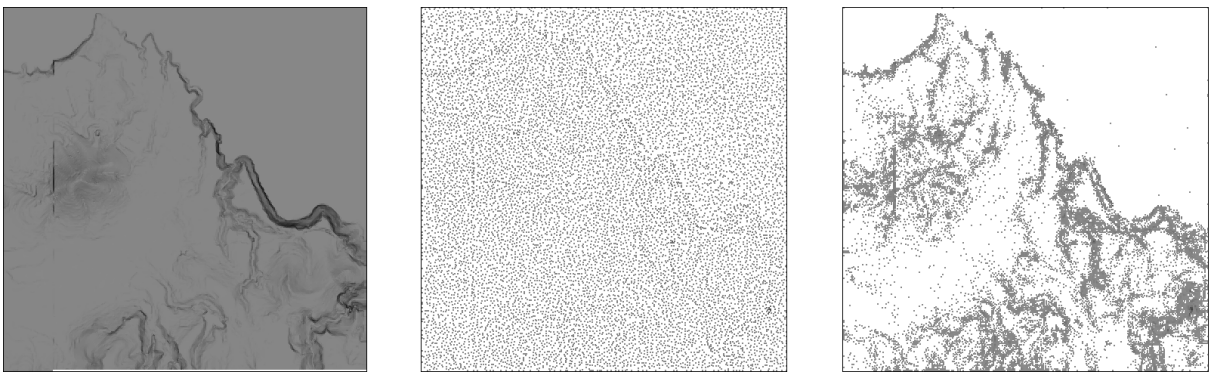


Fig. 14: Points distribution during simplification of dense terrain model (left) with (middle) and without (right) vertex memory. The middle and left picture shows model after 80% simplification (20% of original data).

5 Results

The method has been tested on many models, mostly from GaTech, Cyberware and Avalon depositories. Table 1 shows some fundamental information about models on which we will present our results. All the experiments were done on Intergraph TDZ2000 400MHz Pentium II with 512MB RAM, running on WindowsXP.

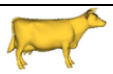
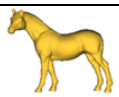
Name	**cow**	**fandisk**	**teeth**	**bunny**	**horse**	**bone**	**terrain**	**dragon**
# vertices	2.905	6.475	29.166	35.947	48.485	60.537	65.829	437.645
# triangles	5.804	12.946	58.328	69.451	96.966	137.072	130.630	871.414
Picture								

Table 1: Models used for presented results.

Table 2 shows the running times of 80% reduction. It is obvious that rapid-flat method (approach without vertex memory) is faster but the resulting mesh contains long and thin triangles. On the other hand the approach with vertex memory produces nicely shaped triangles but the running times are slightly worse.

Name	**cow**	**fandisk**	**teeth**	**bunny**	**horse**	**bone**	**terrain**	**dragon**
Mem	1.244	4.604	11.700	13.304	21.032	26.116	31.728	141.980
No mem	1.160	4.116	10.120	12.320	19.848	24.008	28.872	131.804

Table 2: Reached times [sec] for 80% reduction. Thresholds have been set to mark 15% vertices as extreme.

On Fig. 15 you can see the resulting meshes of both methods for *fandisk* model. However, at most drastical reduction (99% and more) the resulting meshes are similar for both, with and without memory, approaches.

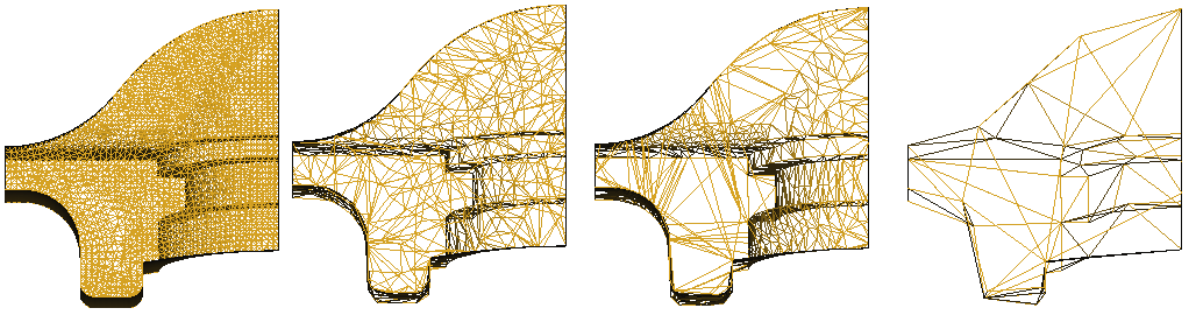


Fig. 15: Example of reduced model. The original mesh (left), 90% reduction with and without vertex memory (two in the middle) and drastical 99% reduction (right).

On Fig. 15 you can see graph of error estimation for several models during simplification process. The models has been simplified from 0% up to 90%. The results are taken from METRO ver. 4.05 [13], using default values (vertex, edge and face sampling enabled, montecarlo sampling, 10times more samples than triangles in a mesh). To have all the values comparable, the METRO results were taken with respect to Dragon model, thus re-computed using following formula (21):

$$E_r = E_M \frac{V_c}{V_{\max}}, \quad (21)$$

where E_M is the value evaluated by METRO, V_c is the number of vertices of current model in certain level of detail and V_{max} is the number of vertices of Dragon model, which is the maximum number of vertices for certain LOD.

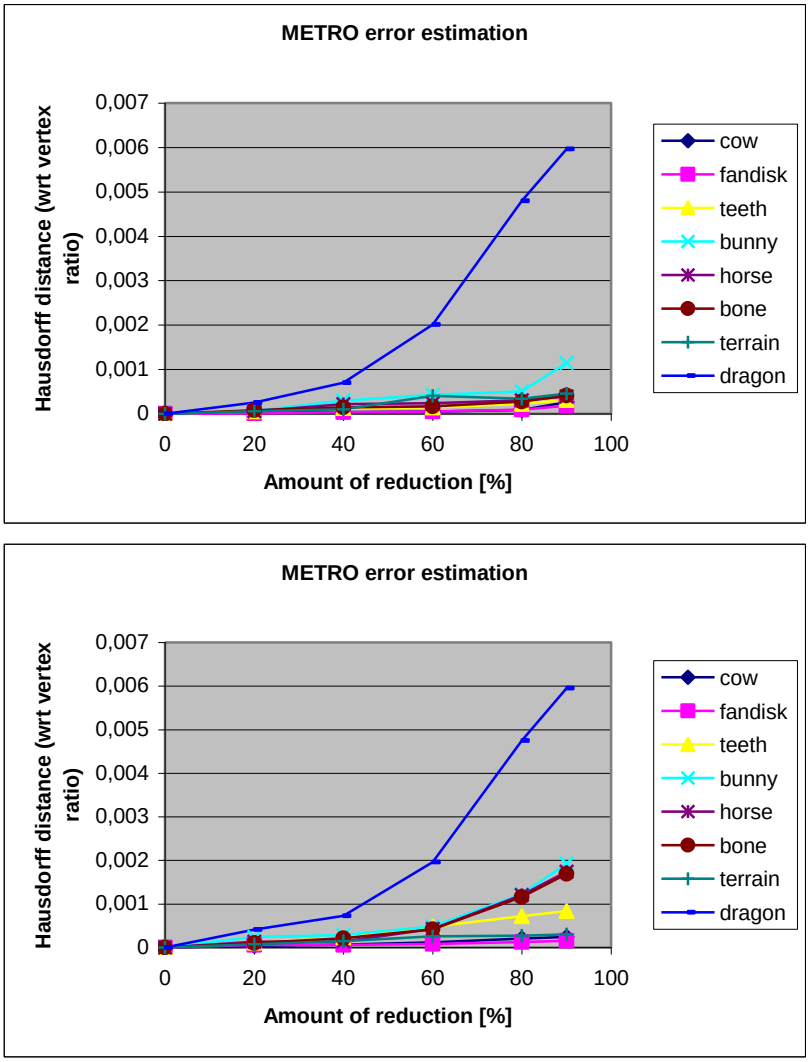


Fig. 16: Aproximation error for certain LOD for approaches with (upper) and without (lower) memory.

It's obvious that memoryless approach gives worse result in meaning of the Hausdorff distance. However, vertices distribution follows ones assumption that flat regions needs to be built from much less number of vertices than rugged surface. Here is noticeable difference between geometrical and perceptive evaluation of the approximation quality.

Also the oversampled models such as dragon, bunny and bone have error values higher than other datasets. Although, the values are higher than other simplification methods, it must be pointed out that METRO computes the error based on Hausdorff distance which is not considered during a simplification in this case. The main goal of presented algorithm is to keep the similarity of appearance. However, the geometrical error is also important in mesh simplification to be able to compare the results with other methods. In Table 3 there are outputs of METRO in detail for cow, bunny and dragon models.

name	reduction	vertices	faces	area	bbox diag.	H-dist
dragon	0%	437645	871414	0.1452	0.266905	0.005964
	90%	41603	81808	0.1446	0.266801	
bunny	0%	35947	69451	0.1143	0.250246	0.022444
	90%	3824	5368	0.1125	0.249250	
cow	0%	2905	5804	2.1802	1.271114	0.032040
	90%	391	776	2.0851	1.267350	

Table 3: METRO details for chosen models.

6 Conclusion

A new approach for triangular mesh simplification with respect to similarity of appearance was presented. This original method is based on vertex importance evaluation to select the least important vertex to be removed from the mesh. This evaluation uses vertex average normal vector which lowest values concern specific model features to be kept in approximations. Simplification itself is performed as an edge collapse where new vertex position is evaluated with respect to supposed surface of the original object given by the endpoints of the edge, the normal vector at these points and opposite corners of adjacent triangles.
We showed that geometrical error doesn't have to be the only criterion of approximation quality and that a visual appearance can lead to opposite observation. This could be quite important in application such as computer games, 3DTV and other multimedia where mathematical precision is not a principal value. Conversely, preserving main visual features is more relevant.

References:

[1] Ciampalini A., Cignoni P., Montani C., Scopigno R., Multiresolution decimation based on global error. *Technical Report CNUCE: C96021*, Istituto per l'Elaborazione dell'Informazione - Condsiglio Nazionale delle Richere, Pisa, ITALY, July 1996.

[2] Franc M., Skala V., Triangular Mesh Decimation In Parallel Environment. *EUROGRAPHICS Workshop on Computer Graphics and Visualization 2001*, Girona, Spain, pp.39-52, ISBN 84-8458-025-3.

[3] Garland M., Multiresolution Modeling: Survey & Future Opportunities. *Eurographics '99, State of the Art Report*. 1999.

[4] Garland M., Heckbert P.S., Surface Simplification Using Quadratic Error Metrics. *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 209-216, 1997.

[5] Hoppe H., New quadric metric for simplifying meshes with appearance attributes. In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 59--66. IEEE, October 1999. ISBN 0-7803-5897-X. Held in San Francisco, California.

[6] Hoppe H., Progressive meshes. *In Computer Graphics Proceedings, Annual Conference Series, 1996* (*ACM SIGGRAPH '96 Proceedings*) , pages 99-108, 1996.

[7] Pawasauskas J., Generalized Unstructured Decimation. *Advanced Topics in Computer Graphics - CS563*, March 18, 1997.

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

[9] Rektorys K. and at al., *Přehled užité matematiky*. Nakladatelství Prometheus, Praha, ISBN 80 85849 72 0, 1995.

[10] Shaffer E., Garland M., Efficient Adaptive Simplification of Massive Meshes. *IEEE Visualization*, 2001.

[11] Surazhsky, T., Magid, E., Soldea, O., Elber, G., Rivlin, E., A comparison of Gaussian and mean curvatures estimation methods on triangular meshes. *IEEE International Conference on Robotics & Automation*, 2003.

[12] Barrera T., Hast A., Bengtsson E., Surface Construction with Near Least Square Acceleration based on Vertex Normals on Triangular Meshes, *Sigrad 2002*, pp. 43-48.

[13] Cignoni P., Rocchini C., Scopigno R., Metro: measuring error on simplified surfaces. *Computer Graphics Forum, Blackwell Publishers*, vol. 17(2), June 1998, pp 167-174