Progressive Polygonal Meshes

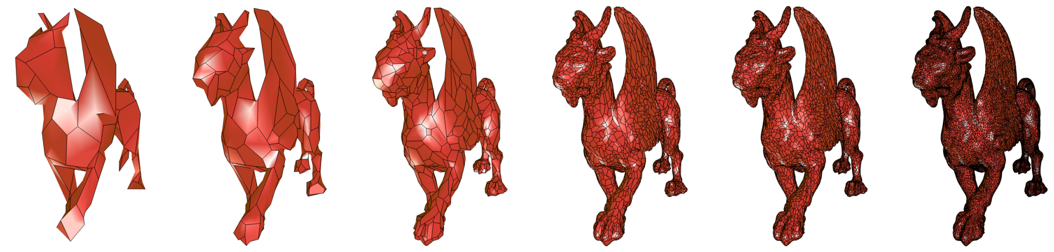
Stefan Gumhold

WSI-2004-4
ISSN 0946-3852

27th July 2004

Wilhelm-Schickard-Institut für Informatik
Graphisch-Interaktive Systeme
Sand 14
D-72076 Tübingen
Tel.: +49-7071-2975463
Fax: +49-7071-295466

e-mail: gumhold@uni-tuebingen.de



a)	b)	c)	d)	e)	f)
$v = 263$	$v = 642$	$v = 2095$	$v = 12182$	$v = 22919$	$v = 45218$
$n = 0$	$n = 581$	$n = 2820$	$n = 19280$	$n = 38667$	$n = 93056$

Figure 1: Polygonal progressive representation of the feline model.

Abstract

In this work we build a progressive polygonal mesh based on face clustering. The basic simplification operations are the edge-removal and the edge-join operations. There is no need to tessellate non-triangular models as the proposed representation is fully polygonal. We investigate three different error measures to build the progressive representation based on a priority queue based face clustering algorithm.

The progressive polygonal mesh representation is built with the inverse refinement operations – the edge-insert and edge-split operations, which can be implemented very efficiently within a newly proposed half-edge data structure. We suggest a progressive file format that allows to read a polygonal model faster than with the standard binary file formats. Finally, we extend the proposed half-edge data structure to a progressive representation that does not consume any further storage space and allows for very fast changes in the mesh resolution.

Contents

1 Introduction

The success of surface representations based on polygonal meshes is due to the large number of available processing tools. Simplification is a very important tool for the efficient processing of large meshes. Most state of the art simplification techniques are based on the mesh decimation technique, that successively applies atomic decimation operations to the original mesh. Among the most important decimation operations are vertex-removal, edge-collapse and triangle-collapse.

A typical simplification algorithm sorts all possible decimation operations into a priority queue, with a priority, that estimates the approximation error caused by the decimation operation. In the decimation loop the simplification algorithm greedily chooses the next best decimation operation in the order given by the priority queue. After the best operation has been performed, the error measure has to be re-evaluated for all remaining decimation operations, which are in the affected neighborhood of the performed operation. The position of the re-evaluated operations in the priority queue is updated before the decimation loop continues with the next best operation until a user specified target mesh size or maximum approximation error is reached.

One important application of mesh decimation is the creation of a progressive representation, that allows for incremental transmission over the internet and the selection of the resolution of a model. To create a progressive representation the simplification algorithm is run as far as possible in order to create the coarsest possible approximation, which is called the *base model*. During simplification the inverse of the decimation operations, i.e. the refinement operations, are stored. The refinement operation of vertex-removal is vertex-insertion and of edge- and triangle-collapse the vertex-split operation. The progressive representation finally consists of the coarse base model together with the refinement operations given in the opposite order, in which they have been recorded during simplification.

The simplification paradigm has also been used for several other applications. Among these are the construction of hierarchical mesh representations that allow for the extraction of meshes that can locally adapt their resolution, incremental computation of a surface parameterization and the clustering of the model into patches. The latter is often used to create a texture atlas of a given model. One important efficiency aspect for the creation of a texture atlas is that the mesh has to be partitioned into non-overlapping patches. This is typically hard to achieve with the previously mentioned simplification operations. Instead are all the patch generation approaches based on face growing or face clustering.

Hierarchical face clustering is very similar to the previously introduced mesh decimation approach, if viewed in the dual domain. The dual graph of a polygonal mesh consists of a node for each face and a face for each vertex of the original mesh. The dual graph has edges between the nodes that correspond to edge-adjacent faces of the original mesh. The original mesh is also called primal as opposed to the dual graph. Collapsing an edge of the dual graph corresponds to the merging of the faces corresponding to the end nodes of the edge. With this mapping between primal and dual graph the hierarchical face clustering algorithm decimates the dual graph with edge-collapse operations.

Non of the existing face clustering approaches allow to create a progressive mesh representation. The existing approaches are either not progressive or have no suitable surface definition in the primal domain. In this work we combine the advantages of hierarchical face clustering with progressive mesh representations that are suitable for rendering. Figure 1 shows an example of the new progressive polygonal representation of the feline model. The progressive representation is based on a surface definition in the primal domain and a simplification algorithm that ensures a valid surface throughout the simplification process, such that the resulting progressive representation can always be visualized. To better understand what happens in the primal domain, where also the approximation error is measured, we interpret the dual edge collapse in the primal domain. This leads to the *edge-removal* operation and its inverse the *edge-insert* operation. During primal edge-collapse simplification, each edge-collapse eliminates the two triangles adjacent to the collapsed edge. This corresponds in a dual edge-collapse to the removal of a vertex of valence two in the primal domain, what is achieved by choining the two edges incident upon the valence two vertex into one. As in the dual edge collapse not always two primal vertices are removed, we consider the *edge-join* operation separately from the edge-removal. The inverse of the edge-join is the *edge-split* operation, which is used in the polygonal progressive model to insert vertices.

One very interesting property of the proposed polygonal progressive representation is the simplicity of its refinement operations. In section 3 we describe how to implement the refinement operations extremely efficiently with a half-edge based data structure. Based on the polygonal progressive representation we propose a new progressive file-format and a new progressive in-core representation for polygonal meshes that are superior to existing formats.

The remainder of the paper is structured as follows. Before the description of the new simplification and refinement operations in section 3 related work is discussed in the next section. In section 4 we introduce the used surface definition necessary to render non planar polygons and to measure error between approximating and original surface. The simplification algorithm is discussed in section 5 and results are given in section 6.

2 Related Work

Mesh Simplification: There is a vast amount of related work on mesh simplification and we refer to some good surveys [10, 5, 8]. Nearly all of the work is on purely triangular mesh. We found only the work of Ramsey et al.[17], where also polygonal meshes are simplified with primal edge- or half-edge-collapse. A very simple error measure based on normal derivation together with an independent set selection technique are used. The proposed face clustering algorithm uses a priority queue as proposed by Klein and Krämer [15]. We implemented three different error metrics: a dual quadric that measures the average distance of points to a best fit plane as proposed by Garland et al. [7], a primal quadric based metric similar but different to the original one from Garland and Heckbert [6] and a one-sided Hausdorff distance, which is close to the one proposed by Klein et al. [14]. In the results section it will turn out that our modification to the primal quadrics give by far the best trade-off between approximation quality and speed.

Face Growing: Kalvin and Taylor [12] grow superfaces from seed triangles, which can be chosen in different ways. They grow each superface by adding faces on the boundary until an user specified planarity limit is reached. They continue growing superfaces until no more triangles are left. Next the boundaries between the superfaces are straightened and a polygonal mesh is created, whose polygons are triangulated with a minimum number of triangles. Compared to our approach is the superface technique not hierarchical.

Sorkine et al.[20] follow a similar face growing strategy to find patches, which can be flattened with bounded distortion. The patches are then used for parameterization.

Face Clustering: The dual edge-collapse has been proposed first by Inoue et el. [11], who form rectangular face clusters on CAD-models with large planar parts. The face clusters are used to remesh the surface with quadrilateral faces.

Garland et al. [7] use the dual edge collapse to construct a hierarchy of nested face clusters. These are used to speed up radiosity computations but no proper surface representation is proposed to render a coarse approximation. The cluster merging criterion is chosen to minimized the average squared distance of the vertices in the cluster to the plane that minimizes this planarity criterion. We tried to use this error measure for our purposes but with limited success as shown in the results section.

Sheffer [19] uses the face clustering technique to simplify CAD models. She follows the priority queue based decimation paradigm and uses several heuristically weighted criteria to order the clustering operations. Among them are the region curvature and optimization criteria for the patch boundary shape. Like Inoue et al. does Sheffer use the clustering as preprocessing to remeshing.

Sander et al. [18] and Lévy et al. [16] use the face clustering approach to split a given input model into disjoined patches, which are used for texture mapping. The clustering is steered by measuring the distortion in the patch parameterizations. After the clustering stage the patch boundaries are straightened. Sander et al. do a second simplification of the original mesh based on primal edge-collapse, where they respect previously built patches.

3 Progressive Polygonal Mesh Data Structure

This section describes the proposed progressive representation and the used data structure, which is based on a combination of half-edge and winged-edge data structure as described in 3.1. This combined data structure is denoted as the *twinned half-edge* data structure. In section 3.2 the basic coarsening and refinement operations are discussed before the progressive representation is introduced in 3.3.

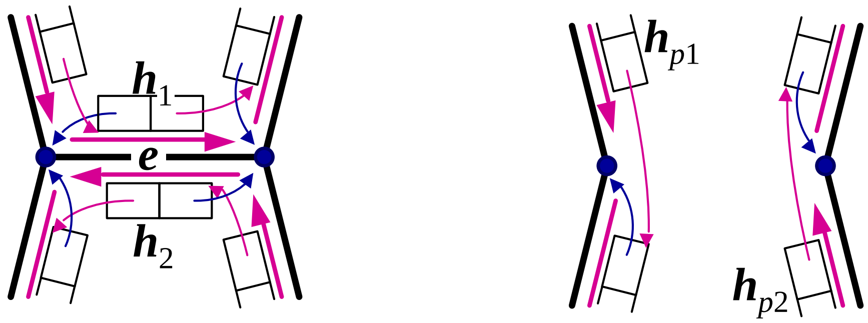
3.1 Single Resolution Polygonal Data Structure

Both the half-edge data structure [21, 13, 4] and the winged-edge data structure [2] have their advantages and disadvantages in representing the connectivity of polygonal meshes. In this work we combine both approaches to the so-called *twinned half-edge* data structure. Figure 2 illustrates the two fields stored with each half-edge ***h***: a reference ***h*.nxt** to the next half-edge in the same face (magenta arrows) and a reference ***h*.origin** to the origin vertex (blue arrows). The important step towards the winged-edge data structure is to rearrange the half-edges such that the two half-edges composing one edge are in successive order in the list of half-edges. This allows to compute the inverse half-edge ***h*.inv** with a binary XOR 1 on the half-edge index and makes the explicit storage of the inverse pointer unnecessary. The re-ordered list of half-edges can also be interpreted as a list of edges. Each edge is formed by two half-edges and identical to a winged-edge without explicit storage of the references ***h*.prv** to previous half-edges. ***h*.prv** is implemented with a loop through the face along the next pointers. One can efficiently transform half-edge indices into edge-indices and vice versa by dividing/multiplying with two.

In order to later on store quadrics with each face, we extend each half-edge by a face index and construct an additional list of faces. This additional storage space is not needed for the progressive representation.

3.2 Coarsening and Refinement Operations

The proposed progressive representation is based on two primitive operations and their inverse: The edge-removal and edge-insertion operations as illustrated in Figure 2 and the edge-join and edge-split as shown in Figure 3. The edge-removal operation is used by the simplification algorithm to



$\boxed{\textit{removeEdge}(\boldsymbol{e})}$

```
e.h_1.prv.nxt= e.h_2.nxt
e.h_2.prv.nxt= e.h_1.nxt
```

a)

$\boxed{\textit{insertEdge}(\boldsymbol{h}_{p1}, \boldsymbol{h}_{p2}, \boldsymbol{e})}$

```
e.h_1.nxt     = h_p2.nxt
e.h_1.origin  = h_p1.nxt.origin
e.h_2.nxt     = h_p1.nxt
e.h_2.origin  = h_p2.nxt.origin
h_p1.nxt      = e.h_1
h_p2.nxt      = e.h_2
```

b)

Figure 2: a) before edge-removal operation, b) after edge-removal and before edge-insert operation.

merge two faces. It simply eliminates the edge from the data structure and corresponds to an edge-collapse in the dual graph. Its inverse – the edge-insert operation – splits the previously merged face by re-introducing the removed edge. The edge-join operation is triggered by the edge-removal, whenever a valence two vertex arises. The two edges incident upon the valence two vertex are joined into one edge by eliminating the valence two vertex from the data structure. Its inverse – the edge-split operation – splits the joined edge and re-introduces the previously eliminated vertex to the data structure.

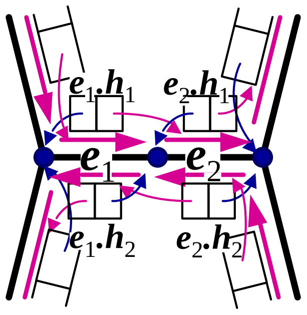
The edge-insert operation is uniquely defined by the two half-edges $\boldsymbol{h}_{p1}$ and $\boldsymbol{h}_{p2}$ preceding the half-edges of the to be inserted edge $\boldsymbol{e}$ (compare Figure 2 b). These are called the *anchor* half-edges. As the anchor half-edges have to be in the same face, the second anchor can optionally be defined as the number i of edges that separate it from the first anchor. The edge-insert operation is abbreviated as

$$\mathrm{I}(\boldsymbol{h}_{p1}, \boldsymbol{h}_{p2}) \qquad \text{or} \qquad \mathrm{I}(\boldsymbol{h}_{p1}, i).$$

The edge-split operation only needs one anchor $\boldsymbol{h}$ (see Figure 3 b) and the geometric location (x, y, z) of the introduced vertex $\boldsymbol{v}$. The short notation for a split operation is

$$\mathrm{S}(\boldsymbol{h}, x, y, z).$$

The updates in the twinned half-edge data structure are very simple for all the coarsening and refinement operations. The necessary commands are shown in the corresponding figures. For the removal of an edge $\boldsymbol{e}$ one only needs to relink the next pointers of the previous half-edges. The method implementing the edge-insert operation in turn has the anchor half-edges and the inserted edge as parameters. It inserts the half-edges of the new edge into the face loop of half-edges, which



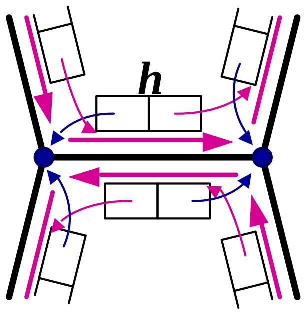
$\boxed{\mathit{joinEdges}(\boldsymbol{e}_1, \boldsymbol{e}_2)}$

$\boldsymbol{e}_1.\boldsymbol{h}_1.\mathbf{nxt} \;\; = \boldsymbol{e}_1.\boldsymbol{h}_1.\mathbf{nxt}.\mathbf{nxt}$

$\boldsymbol{e}_1.\boldsymbol{h}_2.\mathbf{origin} = \boldsymbol{e}_2.\boldsymbol{h}_2.\mathbf{origin}$

$\boldsymbol{e}_2.\boldsymbol{h}_2.\mathbf{prv}.\mathbf{nxt} = \boldsymbol{e}_1.\boldsymbol{h}_2$

a)



$\boxed{\mathit{splitEdge}(\boldsymbol{h}, \boldsymbol{e}, \boldsymbol{v})}$

$\boldsymbol{e}.\boldsymbol{h}_1.\mathbf{nxt} \;\; = \boldsymbol{h}.\mathbf{nxt}$

$\boldsymbol{e}.\boldsymbol{h}_1.\mathbf{origin} = \boldsymbol{v}$

$\boldsymbol{e}.\boldsymbol{h}_2.\mathbf{nxt} \;\; = \boldsymbol{h}.\mathbf{inv}$

$\boldsymbol{e}.\boldsymbol{h}_2.\mathbf{origin} = \boldsymbol{h}.\mathbf{inv}.\mathbf{origin}$

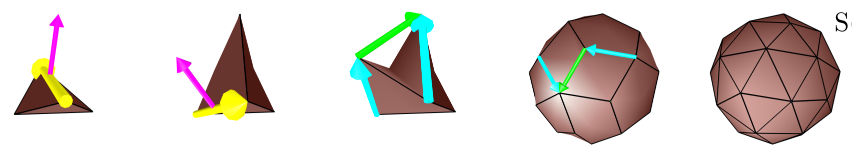
$\boldsymbol{h}.\mathbf{inv}.\mathbf{origin} = \boldsymbol{v}$

$\boldsymbol{h}.\mathbf{inv}.\mathbf{prv}.\mathbf{nxt} = \boldsymbol{e}.\boldsymbol{h}_2$

$\boldsymbol{h}.\mathbf{nxt} \;\; = \boldsymbol{e}.\boldsymbol{h}_1$

b)

Figure 3: a) before edge-join operation, b) after edge-join and before edge-split operation.



$\mathrm{S}(\boldsymbol{h}_3, 0.3, -0.9, 0)$
$\mathrm{S}(\boldsymbol{h}_{10}, 0.5, -0.2, 0.8)$
$\mathrm{I}(\boldsymbol{h}_{10}, \boldsymbol{h}_3)$
$\mathrm{S}(\boldsymbol{h}_0, 0, 0, -1)$
$\vdots$

a) $\mathcal{M}_0$ b) $\mathcal{M}_1$ c) $\mathcal{M}_2$ d) $\mathcal{M}_{55}$ e) $\mathcal{M}_{114}$ f) first ops

Figure 4: An example for the progressive representation of a small sphere, where in a)-d) the next refinement operation is illustrated. e) shows the original mesh corresponding to $\mathcal{M}_{114}$ and f) lists the first refinement operations

is split into two loops. Also the implementations of edge-join and edge-split are straight forward. In the edge-split operation the origin vertex of one anchor half-edge is replaced by the introduced vertex. This would not be necessary in a "not twinned" half-edge data structure. In a primal vertex-split the origin vertices of all triangles in one of the resulting vertex fans has to be changed, what is far less efficient as the proposed refinement operations.

3.3 Progressive Representation

The proposed polygonal progressive representation consists of the base model $\mathcal{M}_0$ and the sequence of refinement operations. As each refinement operation increases the number of edges by one, the refined meshes $\mathcal{M}_n$ are enumerated by the number of introduced edges n. Figure 4 shows an example of the progressive representation of a small sphere. In a) we see the base mesh $\mathcal{M}_0$

and an illustration of the first edge-split operation $\mathrm{S}(\boldsymbol{h}, x, y, z)$, where the yellow arrow shows the half-edge $\boldsymbol{h}$ and the magenta arrow points to (x, y, z). In b) we see the result of the first split, the mesh $\mathcal{M}_1$ with another split to follow. In c) the next operation is an edge-insert $\mathrm{I}(\boldsymbol{h}_{p1}, \boldsymbol{h}_{p2})$ with the half-edges $\boldsymbol{h}_{p1}$ and $\boldsymbol{h}_{p2}$ shown as cyan arrows and the newly introduced edge as green arrow. d) shows $\mathcal{M}_{55}$ and e) the original mesh corresponding to $\mathcal{M}_{114}$. On the right of the figure the first refinement operations are given.

The ordering of the mesh elements is arbitrary in the base model, but the remaining elements are given in the order, in which they are inserted to the progressive representation. A counter v for the current number of vertices and a counter e for the current number of edges is kept. These counters are used to determine the index of newly introduced vertices and edges and in the end hold the total number of vertices and edges.

3.4 Binary Progressive File Format

A first application of the progressive polygonal representation is a simple binary file format, that contains the base mesh together with the refinement operations. The base mesh is stored as a list of vertex coordinate triples and a list of half-edges consisting of two indices each, the index of the origin vertex and the index of the next half-edge in the same face.

The refinement operations follow with two 32-bit indices $\boldsymbol{h}_1, \boldsymbol{h}_2$ for each edge-insert operation $\mathrm{I}(\boldsymbol{h}_1, \boldsymbol{h}_2)$ and one 32-bit index $\boldsymbol{h}$ plus three 32-bit floats x, y, z for or each edge-split operation $\mathrm{S}(\boldsymbol{h}, x, y, z)$. The indices and coordinates are simply concatenated. In order to be able to distinguish insert and split operations, the index $\boldsymbol{h}$ of the split operation is marked by replacing it with the negative index $(-1 - \boldsymbol{h})$. This is possible because all indices are zero or positive and therefore a negative one can always be recognized.

The size of the binary format can be easily computed to be $2v + 2e + v_{\mathrm{bas}} + 22_{\mathrm{bas}}$, with the number of base model vertices v_{bas} and edges e_{bas}. As the base model is typically very small, only $2v + 2e$ indices or floats are consumed, i.e. v indices less than in any face-based mesh format like ply, obj or vrml.

The second advantage of the progressive binary format is that it implicitly stores all the connectivity information of the mesh. After reading one of the face-based file formats the inverse pointers of the half-edges have to be computed by some sorting strategy. An optimized linear time bucket-sorting algorithm for this task was nearly three times slower than the construction from the progressive binary format.

3.5 In-Core Progressive Representation

The second application of the proposed format is an in-core progressive half-edge data structure. The goal is to alter the mesh resolution and always have an active half-edge data structure in-core, i.e. one extends the API of the twinned half-edge data structure by a method that adjusts the mesh resolution to n.

Instead of storing the progressive representation in a separate memory block, one can store it inside the half-edge data structure without any additional storage space consumption. Figure 5 illustrates this interleaved memory layout. Suppose that resolution n is adjusted. Then the first $(e_{\mathrm{bas}} + n)$ edges represent the twinned half-edge data structure of $\mathcal{M}_n$. In the remaining half-edge records the information of the edge-insert and edge-split operations is kept, where no vertex locations have to be stored with the edge-split operations as the vertices are kept in a separate vertex

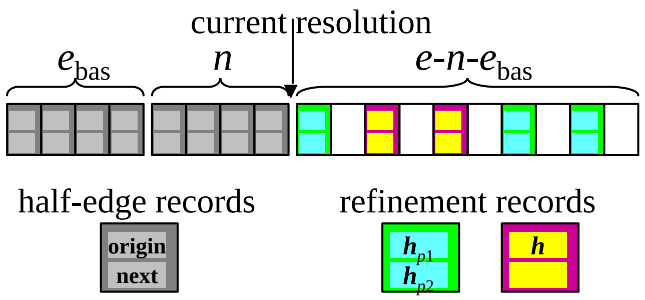


Figure 5: Illustration of the interleaved progressive in-core half-edge data structure. Each block corresponds to one half-edge and two successive blocks to one edge.

list. Therefore, only the anchor half-edges have to be stored and only half of the available storage space is used for this.

To increase the mesh resolution, the refinement operation stored in edge $(e_{\text{bas}}+n+1)$ is applied and overwritten by the newly constructed edge. To decrease the resolution the edge with the index $(e_{\text{bas}}+n)$ has to be eliminated. If the valence of the origin vertex is two (which is the case, iff $\boldsymbol{h}.\mathbf{inv}.\mathbf{nxt}.\mathbf{inv}.\mathbf{nxt} = \boldsymbol{h}$), a join operation has to be performed and the number of vertices decreased, otherwise a remove-edge operation is performed. In both case the parameters of the corresponding coarsening operation overwrite the two half-edge records of the removed edge.

Experiments on the previously used P4 with 2.4 GHz test machine show that with this approach about 1.5 million edges can be inserted or removed per second.

4 Polygonal Surface Definition

In order to allow for efficient rendering and to measure the approximation error an essential ingredient to the polygonal progressive mesh representation is a proper definition of the surface spanned by non planar polygons. There are several possible choices: minimal area surfaces, polygonal subdivision surfaces, tesselation, etc.

There are three requirements that we think should be fulfilled by a practical surface definition: first should the surface definition be local. If the definition is not local like as for example in the case of subdivision surfaces, the measuring of the approximation error and potential view-dependent refinement applications will be much more difficult. Secondly, the surface definition has to allow for efficient rendering. And thirdly must the definition be symmetric with relation to the vertices of the polygon. Otherwise it makes not much sense to use polygons.

We found that the best compromise for the three requirements is a star shaped triangulation of the polygons around a face center vertex, that is computed from the polygonal mesh. A triangulation with a minimum number of triangles as used by Kalvin and Taylor [12] is not symmetric with relation to the polygon vertices and would demand for extra information. We want to emphasize here that the face center vertex and the star shaped tessellation can be efficiently computed on the fly and do not contributed to the storage space consumed by the progressive polygonal representation.

We experimented quite a lot with different definitions for the face center location, but it turned out to be most robust and also fastest to simply use the center of mass of the vertices in the tesselated polygon. This definition makes the surface definition only dependent on the vertices of the polygon

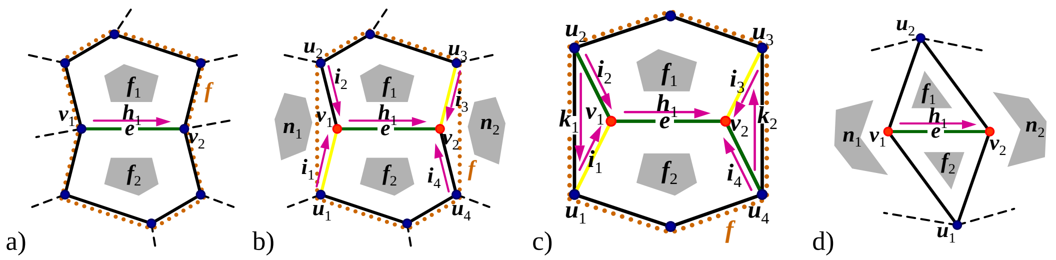


Figure 6: Collection of the different cases of triggered edge-join operations. The resulting merged face $\boldsymbol{f}$ is outlined dotted in brown. a) no triggered operations, b) two triggered joins, c) triggered joins degenerate neighbor faces, d) joins degenerate merged face.

and fulfills all three requirements. The rendering performance could be slightly improved because of the two additional triangles per polygon, but the star shaped triangulation allows for the use of triangle fans, which can be rendered faster than independent triangles. Furthermore, in the application of progressive transmission over the internet is the rendering cost negligible.

5 Polygonal Mesh Simplification

For the construction of the progressive polygonal mesh representation we follow the priority queue based mesh decimation approach. Three different error measures have been investigated to sort the edge-removal operations into the priority queue. These are discussed in section 5.3. Important is that the error measure is evaluated on the mesh where not only the edge-removal has been performed but also all induced edge-join operations. Before the next best edge-removal operation is performed during the decimation loop, we check, if it leads to degenerate topological or geometrical constellations. If yes, the edge-removal operation is skipped. The validity tests are described in sections 5.1 and 5.2. If the edge-removal is valid, it and all triggered edge-join operations are performed and the anchor half-edges and in case of a edge-join also the index of the removed vertex is recorded for the creation of the progressive representation.

For a clean implementation it is very helpful to distinguish the different cases of triggered edge-join operations and to build a temporary *removal-info* data structure that captures the necessary data. The different cases are shown in Figure 6. The dotted brown outline shows the face $\boldsymbol{f}$ after the removal and triggered join operations. Face $\boldsymbol{f}$ is used to measure the error of the operation. In a) the simplest edge-removal case is shown without triggered edge-join operation. b) shows the cases with one or two triggered edge-joins. The half-edges $\boldsymbol{i}_{1\ldots4}$ are stored in the removal-info or set to -1 where no join is triggered. The neighbor faces $\boldsymbol{n}_{1/2}$ are influenced by the join operations (compare section 5.3.4). Figure 6 c) shows the third case, when the neighbor faces have degree three end degenerate. This case is distinguished in the removal-info by the additional storage of $\boldsymbol{k}_{1/2}$. Case c) is translated into the following sequence of edge-removal and edge-join operations: remove $\boldsymbol{i}_2$, join $\boldsymbol{i}_1$, remove $\boldsymbol{i}_4$, join $\boldsymbol{i}_3$, remove $\boldsymbol{h}_1$. As the valence of the vertices $\boldsymbol{u}_{1\ldots4}$ is decreased by one, further edge-join operations could be triggered. We avoid this combinatorial explosion by prohibiting the removal of edge $\boldsymbol{e}$ in case c) whenever an edge-join operation would be triggered on an $\boldsymbol{u}_j$. Cases b) and c) can also arise only on the left or right or mixed.

One of our topological constraints says that two faces may not touch in more than one vertex or edge. This rules out the nasty case d) of Figure 6, where also the merged face $\boldsymbol{f}$ would degenerate.

But for that the neighbor faces $\boldsymbol{n}_{1/2}$ would have to touch in the two vertices $\boldsymbol{u}_{1/2}$.

Compared to primal edge-collapse simplification does case a) have no pendant, b) corresponds to the regular case and c) is the case of a degree three vertex incident to one of the triangles that are edge-adjacent to the collapsed edge. This case is typically forbidden and the corresponding edge-collapse discarded. Case d) can arise in the primal approach for triangulated models only in the case of a tetrahedron.

5.1 Topological Constraints

In the presented approach it is assumed that the connectivity of the input mesh fulfills the following properties:

- manifoldness
- the minimal vertex valence is three
- the minimal face degree is three
- no face touches itself
- the intersection of any two faces is empty or connected

If the input mesh is non-manifold it has to be cut into manifold pieces. If the face properties are violated, they have to be enforced by cutting the faces, but in practice we did not encounter any polygonal model, where the face properties were violated.

All properties are preserved during simplification. This will also preserve the genus of the input mesh. The manifoldness cannot be violated by edge-removal or edge-join operations without violating one of the other three properties. The vertex valence and face degree properties are ensured by the triggered join operations and the avoidance of all case c) in Figure 6, where a vertex valence of $\boldsymbol{u}_j$ decreases below three.

Also the face touching properties can be easily enforced. As no vertex of valence two arises, the intersection of two faces cannot have any connected component with more than one edge. Thus the second face property is valid, if two faces do not touch in more than one edge or one vertex and no edge. Only one face flag is necessary to check this. Before the decimation loop the flag is set to false for all faces. To check the second face property we cycle once around the resulting merged face ***f*** with the help of the removal info data structure. At every half-edge we check the flag of the edge-adjacent face. If this is set, we found a violation. Otherwise we mark the flag. Next we cycle around the end vertex of the half-edge and do the same check and mark commands on the not edge-adjacent but vertex-adjacent faces. After the complete cycle we know if the face property has been violated. In anycase we set the changed flags to false again. To check, whether a face touches itself we simply compare the indices of the edge- and vertex-adjacent faces with the current face.

5.2 Geometric Constraints

The only geometric constraint important to rendering is that no normals flip. We implemented the following normal flip test: compute the face center of the resulting merged face ***f*** (compare Figure 6 a-c), compute the least square fit plane to the vertices of face ***f*** and extract its normal as the reference normal, compare the cosines of the angles between the triangle normals of the star shaped tesselation of ***f*** and the reference normal. If one of the cosines is below a user defined threshold, the edge-removal operations is rejected. For all experiments we used a threshold of 0.5.

5.3 Error Measures

5.3.1 Planarity Based Dual Quadric

The first attempt for the error measure was the dual quadrics approach as described in [7]. It uses the squared average distance of all points in a face-cluster to a best fit plane. A similar approach is found in [18] with the difference that the squared average distance is integrated over the triangles of the original patch. We only tried the simpler version as the planarity based error measure does not perform well. This is on the one hand clearly reflected in the results section and on the other hand also intuitively explainable: one can easily show that the dual quadric error measure does not change if an edge-join operation is performed. As the edge-join operations remove the vertices from the mesh, it is hard to believe that the planarity based error measure makes sense. We could also not easily generalize the method to models with border.

5.3.2 Primal Quadric Based Error Measure

The second attempt was much more successful. It uses the primal quadrics to approximate the squared distance function to the original mesh as proposed by [6]. In the original work, that is based on primal edge-collapse, the target vertex of the edge-collapse is placed at the location that minimizes the approximated distance function. The error measure is then taken as the distance function evaluated at the optimized target vertex location. The quadric error metric approximates the distance of the coarse approximation to the original mesh by sampling the coarse mesh at one location – the location of the target vertex.

This idea cannot be directly transferred to the simplification based on edge-removal as here the original vertex locations are preserved. If we evaluate the approximate distance function at an original vertex, this will nearly always result in an error larger than zero, what is wrong as we known that the error is zero at original vertices. Therefore, the approximate distance function is not useful at the original vertex locations.

Instead we argue that the approximate distance function becomes more and more valid the farther we move inside of the merged face ***f*** resulting from the edge-removal operation (compare Figure 6 a-c). We therefore propose to sample the approximate distance function in the face center instead of on any original vertex. The result section proofs that this is a very good choice and it would be interesting to apply this idea also to simplification based on primal half-edge collapse or vertex-removal.

The proposed idea can easily be generalized to meshes with border and sharp creases. Border edges can only be decimated by edge-join operations as we do not allow the removal of border edges. To approximate the distance to the mesh border, we accumulate the border [6] and crease [3] quadrics separately from the surface quadrics. The distance function is then sampled on the edge center of the joined border or crease edge. The final error of an edge-removal operation, which changes the mesh border, is computed as the maximum error of surface- and border-error.

5.3.3 Hausdorff-Distance Based Error Measure

If viewed in terms of the number of vertices are the rate distortion curves for the proposed approach with the primal quadric error metric a bit worse than the results achieved with QSlim. To examine the best achievable result with face clustering we also implemented an error measure that is based on the one-sided Hausdorff distance from the original mesh $\mathcal{M}_\infty$ to the coarse approximation $\mathcal{M}_n$

of resolution n:

$$d_{\mathrm{HD1}}\left(\mathcal{M}_\infty, \mathcal{M}_n\right) = \max_{\mathbf{p}_\infty \in \mathcal{M}_\infty} \min_{\mathbf{p}_n \in \mathcal{M}_n} ||\mathbf{p}_\infty - \mathbf{p}_n||.$$

We approximated the one-side Hausdorff distance in the following parametric way: we sampled $\mathcal{M}_\infty$ on its vertices and on the edge centers, we assigned the sampled vertices to edges and faces in the coarse approximation and computed the error of a merged face ***f*** as the minimal distance of all the samples, that have been assigned to the face and its boundary edges, to the star shaped tessellation of the face ***f***. This modified Hausdorff-distance also tries to keep straight patch boundaries and automatically measures the error at the mesh border. It does not solve the problems resulting from the use of the one-side Hausdorff distance though. For this one could additionally measure the distance of the face and border edge centers to the original model, but we did not implement this.

The sample assignments are updated as follows: In the preprocessing stage we create the samples on the edge centers and assign them to their edges. No vertices are assigned to original faces. After each removal operation, the vertices assigned to the removed edge and the two adjacent faces are unified and all assigned to the resulting face. After each join operation the vertices assigned to the joined edges together with the eliminated vertex are assigned to the resulting edge.

The running time of simplification with the Hausdorff based error measure is quadratic in the size of the simplified model and only suitable for small models. To overcome this problem, we allowed the user to restrict the maximum number of vertices in assignment lists of edges and faces to a constant $a_{\max}$. In order to select the vertices that are kept in an assignment list, we attached an age to the vertices according to their removal from the mesh. The earliest removed vertices were eliminated first from the lists.

With the introduction of the parameter $a_{\max}$ it became possible to also reduce large models with the Hausdorff-based error measure. In the results section we examined, the influence of $a_{\max}$ on the achieved rate distortion curves.

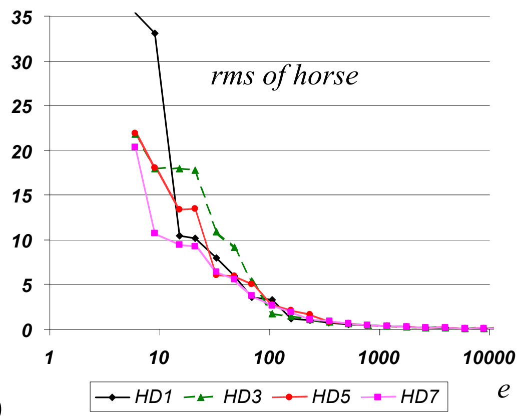
5.3.4 Induced Errors

In the case b) of Figure 6 the geometry of the neighboring faces $\boldsymbol{n}_{1/2}$ are changed. This changes the primal quadric based and the Hausdorff distance based error measures of the affected neighbor faces. We account for that by also evaluating the new value of the error measure of the affected faces and by taking the maximum over all evaluated error measures.

6 Results

All rate distortion curves in this section were measured with the MESH-tool [1]. For this we had to tessellate the polygonal faces as described in section 4. We had problems though for models with border loops such as the Stanford bunny. The measurements were taken on a machine with 512MB and a Pentium 4 with 2.4GHz. All rate-distortion curves are shown in double logarithmic diagrams, what makes the curves close to linear and much easier to read than curves on a singly logarithmic scale.

In the first experiment we examined the influence of the parameter $a_{\max}$, that limits the maximum number of samples considered per edge or face, on the rate-distortion curves for simplification with the Hausdorff-distance. The results are shown in Figure 7 a) for the horse model from the large model archive of Georgia Tech. The increase of $a_{\max}$ does improve the curve slightly. In b) the running time for simplification is plotted over $a_{\max}$. As a compromise between accuracy and running time

a) 

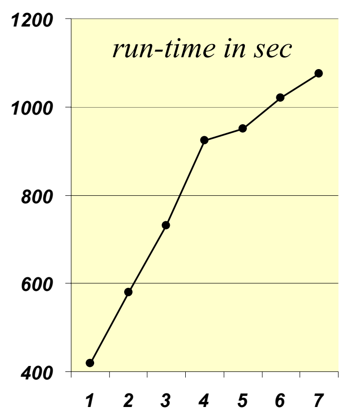
b) 

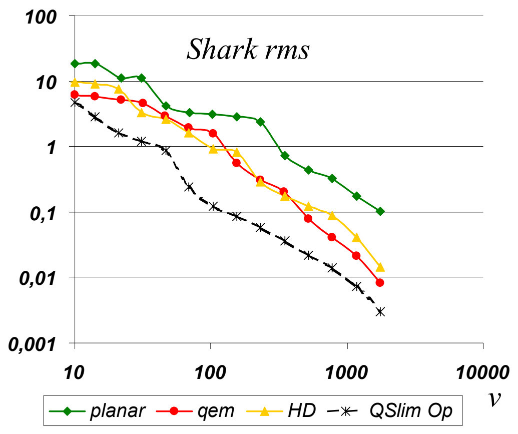
Figure 7: Examination of Hausdorff-distance based root mean squared error rate-distortion curve in dependence on maximum number of assigned samples $a_{\max}$: a) rate-distortion curves for $a_{\max} = 1, 3, 5, 7$, b) running time to simplify model in seconds.

we chose to use the parameter value $a_{\max} = 6$ for all the models larger than 10,000 vertices, where the unrestricted version of the Hausdorff-distance measure was too slow.

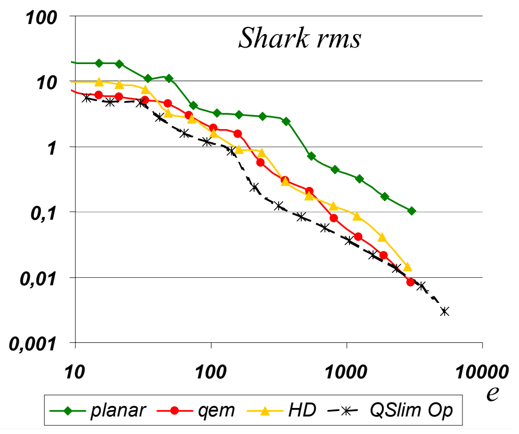
In the second experiment we compared the different error measurements. Figure 8 shows the results for the shark model. In a) and b) the average error is plotted over the number of vertices and the number of edges. We also plotted the results of QSlim with vertex position optimization. The planarity measure is clearly the worst. The primal quadric based measure is comparable with the Hausdorff-distance based measure. Both are worse than QSlim. This is not surprising especially when drawn over the number of vertices. The proposed edge-removal based simplification algorithm does remove a lot of edges before vertices are removed. If the rate-distortion curves are plotted over the number of edges the proposed approach becomes much closer to QSlim. Figure 9 shows another example of the armadillo model. Here also the result of QSlim configured to preserve the original vertex locations is shown. This is approximately the quality that we can reach, what is quite surprising as we do face clustering, which is more difficult as edge-collapse simplification because of the additional constraint of the creation of disjoint patches on the original surface.

A visual analysis of the different proposed error measures is shown in Figure 10 a-c) for the shark model. One can clearly see that the Hausdorff-distance based approach preserves the features to much coarser resolutions. This is also true for the triceratops model, whose progressive representation built with the Hausdorff-distance is shown in d). More examples are shown in Figure 11.

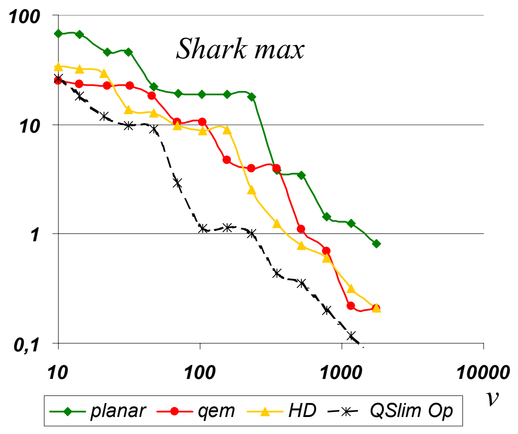
Table 1 tabulates the results of further measurements. The first five models are polygonal, whereas the other five models are purely triangular. Planarity based simplification is a bit slower as the evaluation involves an eigenvalue decomposition. Compared to QSlim is our algorithm about eight times slower. This is on the one hand due to the more complex consistency checks and on the other hand to the larger number of failed validity tests, that are due to the difficulty of disjoint face clustering. The Hausdorff-distance based approach is much slower and was primarily implemented to check how well the proposed primal quadric based error measure is.



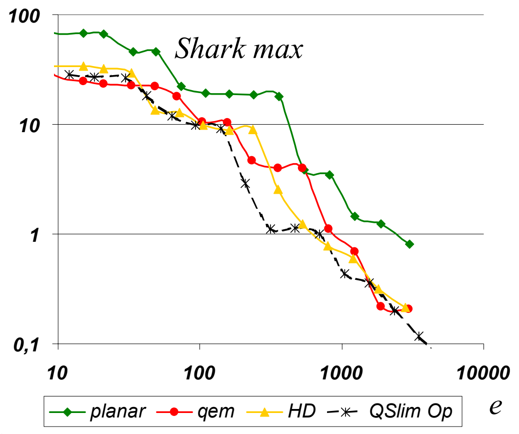
a)



b)



c)

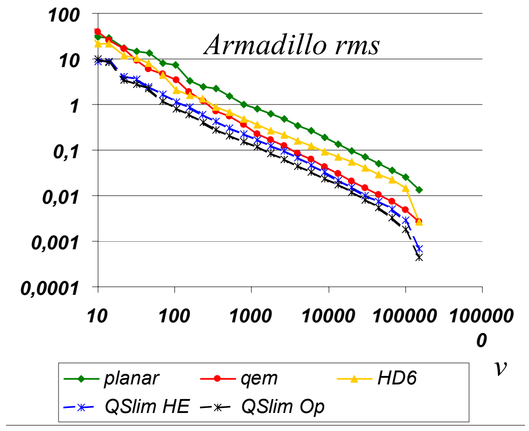


d)

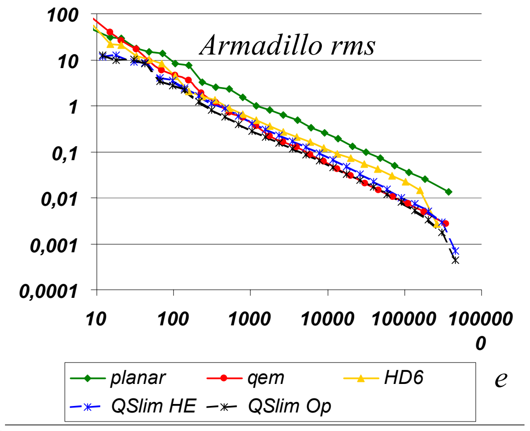
Figure 8: Comparison of rate-distortion curve for shark model over number of vertices a,c) and over number of edges b,d). Root mean squared error in a,b) and maximum two-sided Hausdorff distance in c,d).

7 Conclusion & Future Directions

This work introduces a new progressive mesh representation based on the edge-split and edge-join operations. It allows to handle polygonal meshes without the need for initial triangulation. The method is based on the hierarchical face clustering paradigm and therefore partitions the original mesh hierarchical into disjoint patches. For a lot of applications is this an important advantage. The interpretation of the face clustering operation in the primal domain together with a suitable surface definition for polygons with non planar faces allows to use the new progressive representation also for visualization. We introduced an error measurement based on efficiently computable quadrics, which is fast and achieves very good results compared to our reference implementation of a Hausdorff-distance based error measurement. Compared to triangular mesh simplification methods does the newly proposed method perform a bit worse in speed and a bit worse in approximation



a)



b)

Figure 9: Comparison of root mean squared error rate-distortion curve for armadillo model a) over number of vertices, b) over number of edges.

model	v	e	v_{bas}	e_{bas}	t_{planar}	t_{qem}	t_{HD6}	t_{HD}
cupie	2968	6004	85	156	1	1	-	64
shark	2560	5120	4	6	1	1	-	90
cessna	3737	7650	46	83	1	1	-	87
triceratops	2832	5664	4	6	1	1	-	107
croco	17,332	51,606	260	390	6	7	216	-
cow	2,904	8,706	4	6	1	1	44	120
feline	49,864	149,598	57	86	38	32	1002	-
santa	75,781	227,337	4	6	45	37	1421	-
armadillo	172,974	518,916	4	6	83	89	3225	-
isis	187,644	562,926	4	6	92	97	1213	-

Table 1: Measurements for the used models. From left to right: number of vertices/edges in original and base model, construction time for simplification with planarity measure, primal quadric measure, Hausdorff-measure with maximum of six samples per assignment list and if available not restricted Hausdorff-measure. The first five models are polygonal and the other five triangular.

quality but generates disjoint hierarchical face clusters.

The second advantage of the proposed technique is the very efficient progressive representation, from which we designed a progressive file format that improves disk read times and we presented an extension to the twinned half-edge data structure, that allows to efficiently select a resolution in the progressive representation without any additional storage space consumption.

Some future work [9] already has shown that the new progressive representation can be used to efficiently build a view-dependent polygonal mesh, which allows for very fast and localized changes in the resolution. Further future work will include the compression of progressive polygonal meshes and the combination with texture and normal-maps mapping.

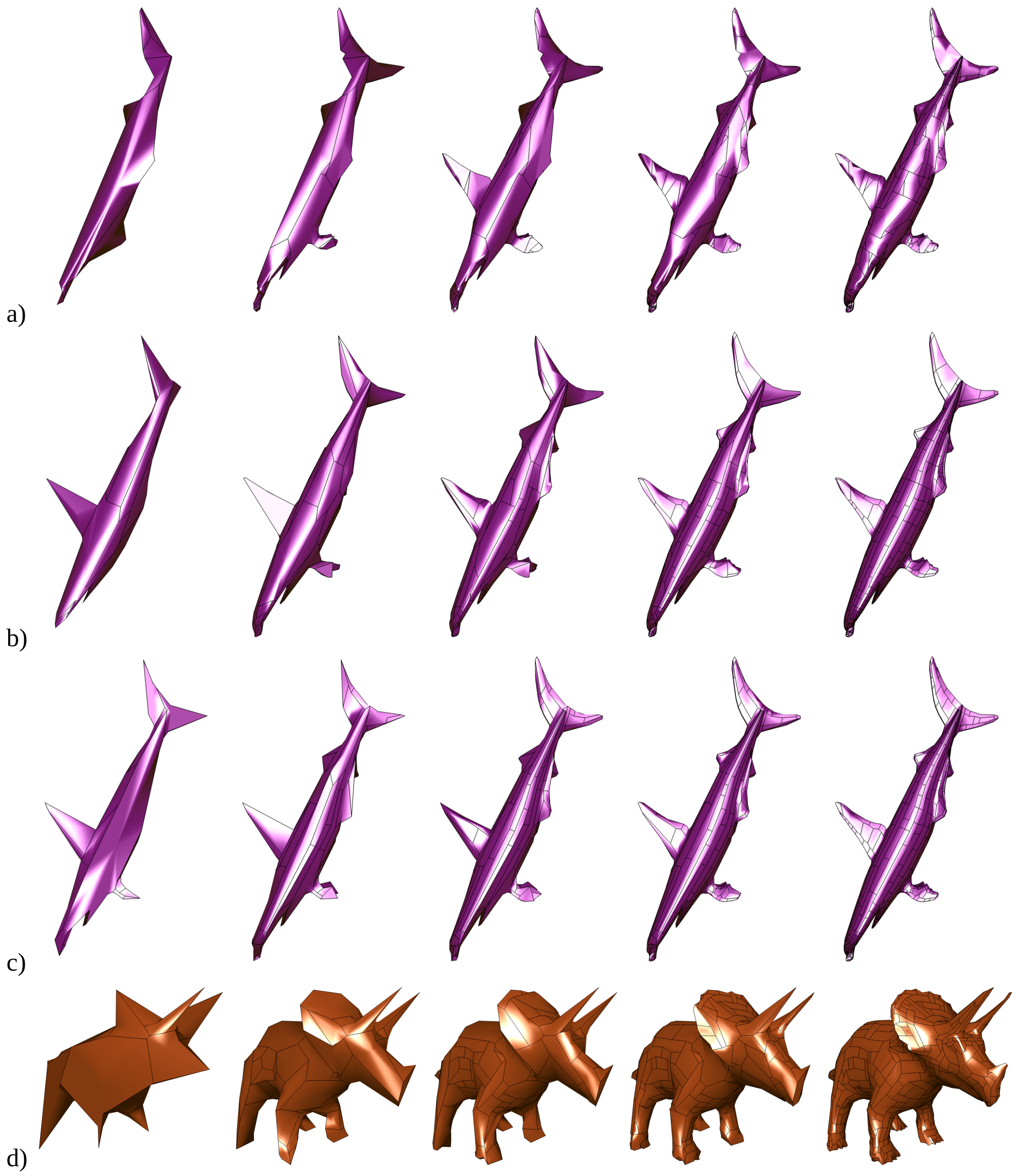


Figure 10: Visual comparison of the investigated error measurements on the shark model for the approximations $\mathcal{M}_{100}, \mathcal{M}_{300}, \mathcal{M}_{600}, \mathcal{M}_{1200}, \mathcal{M}_{2400}$: a) planarity (dual quadrics), b) distance from coarse to fine (primal quadrics), c) distance from fine to coarse (Hausdorff-distance)

d) the same approximation resolutions of the triceratops constructed with unrestricted Hausdorff-distance.

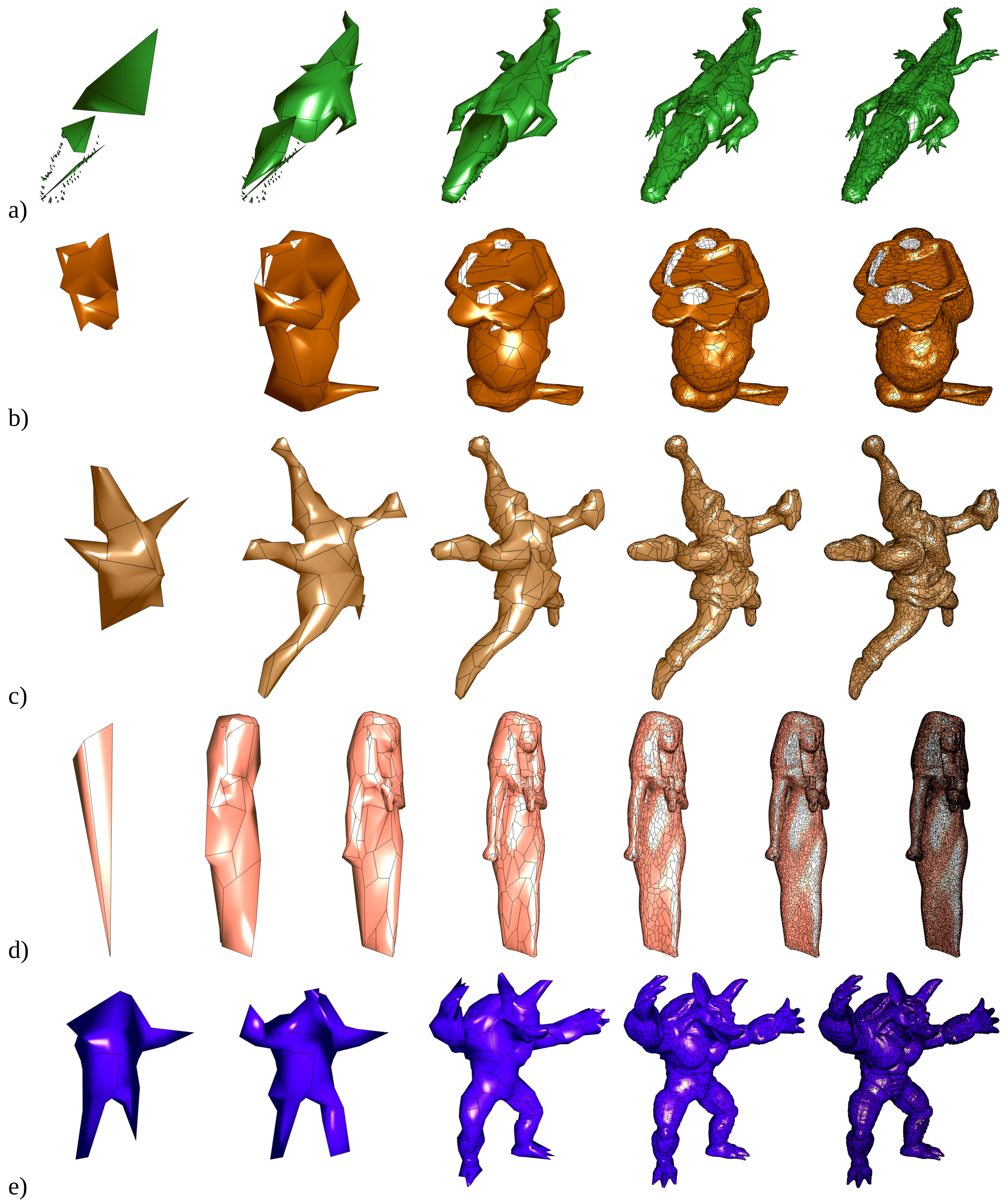


Figure 11: More examples of progressive representations all built with the primal quadrics approach: a) croco model with $n = 0, 100, 500, 3000, 10000$, b) bunny model with $n = 0, 100, 1000, 5000, 20000$, c) santa model with $n = 50, 200, 1000, 5000, 20000$, d) isis model with $n = 0, 100, 500, 2000, 10000, 50000, 200000$, e) armadillo model with $n = 50, 100, 1000, 10000, 100000$

References

[1] N. Aspert, D. Santa-Cruz, and T. Ebrahimi. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, volume I, pages 705 – 708, 2002. `http://mesh.epfl.ch`.

[2] B. Baumgart. A polyhedron representation for computer vision. *AFIPS Nat. Conf. Proc.*, 44:589–596, June 1975.

[3] Pavel Borodin, Stefan Gumhold, and Reinhard Klein. High-quality simplification with generalized pair contractions. In *Proceedings of GraphiCon 2003*, pages 147–154, sep 2003.

[4] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Openmesh – a generic and efficient polygon mesh data structure, 2002.

[5] M. Garland. Multiresolution modeling: Survey & future opportunities, 1999.

[6] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH'97 Conference Proceedings*, pages 209–216, 1997.

[7] M. Garland, A. Willmott, and P. Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, mar 2001.

[8] Craig Gotsman, Stefan Gumhold, and Leif Kobbelt. Simplification and compression of 3d meshes. In Armin Iske, Ewald Quak, and Michael S. Floater, editors, *Tutorials on Multiresolution in Geometric Modelling*, Mathematics and Visualization, pages 319–362. Springer, 2002.

[9] Stefan Gumhold. View-dependent polygonal meshes, 2004. submitted for publication, preprint available online.

[10] P. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms, 1997.

[11] K. Inoue, T. Itoh, A. Yamada, T. Furuhata, and K. Shimada. Clustering large number of faces for 2-dimensional mesh generation. In *Proceedings of the 8th International Meshing Roundtable*, pages 281–292, 1999.

[12] Alan D. Kalvin and Russell H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Comput. Graph. Appl.*, 16(3):64–77, 1996.

[13] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. In *Proceedings of 14th Annual ACM Symposium on Computational Geometry*, 1998.

[14] R. Klein, G. Liebich, and W Straßer. Mesh reduction with error control. In R. Yagel and G. M. Nielson, editors, *IEEE Visualization 96*, pages 311–318. ACM Press, October 1996.

[15] Reinhard Klein and Jorg Krämer. Building multiresolution models for fast interactive visualization. In Wolfgang Straßer, editor, *13th Spring Conference on Computer Graphics*, pages 57–66, 1997.

[16] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *Proceedings of Siggraph '02*, pages 362–371, July 2002.

[17] Shaun D. Ramsey, Martin Bertram, and Charles Hansen. Simplification of arbitrary polyhedral meshes. In *IASTED Computer Graphics and Imaging 2003*, pages 117–222, 2003.

[18] Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. Texture mapping progressive meshes. In *Proceedings of Siggraph '01*, pages 409–416, August 2001.

[19] A. Sheffer. Model simplification for meshing using face clustering. *Computer Aided Design*, 33:925–934, 2001.

[20] Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In Robert Moorhead, Markus Gross, and Kenneth I. Joy, editors, *Proceedings of the 13th IEEE Visualization 2002 Conference (VIS-02)*, pages 355–362, Piscataway, NJ, October 27– November 1 2002. IEEE Computer Society.

[21] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Application*, 5(1):21–40, 1985.