Truly Selective Polygonal Mesh Hierarchies with Error Control

Stefan Gumhold

MPI Informatik, Saarbrücken

Abstract

In this work a new multi-resolution model is proposed for polygonal meshes. It is based on the dual edge collapse, which performs face clustering instead of vertex clustering. The new hierarchical mesh representation combines a truly selective refinement scheme with a strict control of the two-sided Hausdorff distance. The proposed approach allows to build hierarchical meshes directly over non triangulated polygonal models. As most dependencies in the hierarchy are in the form of trees, the resulting representation is very compact and allows for a compressed in-core representation. Furthermore an optimization scheme is proposed for the hierarchy based on variational shape approximation.

Key words: Error Controlled Simplification, View-Dependent Visualization, Polygonal Meshes, Hierarchical Modeling

1 Introduction

The emergence of new 3d scanning technologies formed a demand for the efficient visualization of huge polygonal models. The most important of these models represent beautiful pieces of cultural heritage, prototypes from car, ship and aircraft industry or models extracted from medical 3d images.

To handle such large meshes a variety of efficient processing techniques are necessary. Among the core techniques are compression and simplification. The latter allows to build a hierarchy over the irregular surface tessellation. The mesh hierarchy is used to generate meshes of a varying resolution that is adapted to the current processing task. Among the most important tasks are view-dependent visualization, adaptive global illumination and adaptive simulation approaches.

In order to extract an adaptively refined mesh from the mesh hierarchy, a quality criterion is defined over the surface. The most commonly used criteria are geometric error and for visualization approaches also normal and texture deviation. The goal

of the mesh hierarchy is to generate an adaptively refined mesh with the smallest number of mesh elements that fulfills the quality criterion everywhere. To achieve this, additional attributes are stored within the hierarchy that allow to bound the different quality criteria.

The most natural measure for the geometric error is the Hausdorff distance. It generalizes the Euclidean distance between two points to the distance between two surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$ and is defined as

$$\mathcal{H}_{\mathcal{S}_1 \to \mathcal{S}_2} \stackrel{\text{def}}{=} \max_{\mathbf{p}_1 \in \mathcal{S}_1} \min_{\mathbf{p}_2 \in \mathcal{S}_2} \|\mathbf{p}_1 - \mathbf{p}_2\| . \tag{1}$$

$\mathcal{H}_{\mathcal{S}_1 \to \mathcal{S}_2}$ measures the maximal distance of a point on $\mathcal{S}_1$ to $\mathcal{S}_2$ and as its definition is not symmetric is called the *one-sided Hausdorff distance*. Taking the maximum over the two possible one-sided distances yields the *[two-sided] Hausdorff distance*

$$\mathcal{H}_{\mathcal{S}_1 \leftrightarrow \mathcal{S}_2} \stackrel{\text{def}}{=} \max \left\{ \mathcal{H}_{\mathcal{S}_1 \to \mathcal{S}_2}, \mathcal{H}_{\mathcal{S}_2 \to \mathcal{S}_1} \right\} . \tag{2}$$

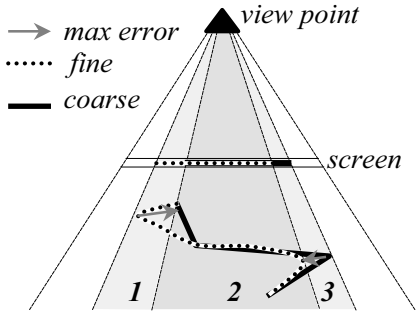


Figure 1. Illustration of the error in screen projection between the fine original mesh (dotted) and the coarse adapted mesh (solid).

The Hausdorff distance can be used to bound the screen space error between a mesh $\mathcal{M}$ and an adaptively refined approximation $\mathcal{M}_{\text{adapt}}$. A bound on $\mathcal{H}_{\mathcal{P}(\mathcal{M}) \to \mathcal{P}\left(\mathcal{M}_{\text{adapt}}\right)}$, where $\mathcal{P}(.)$ is the projection on the screen, ensures that all pixels covered by $\mathcal{M}$ are also covered by $\mathcal{M}_{\text{adapt}}$. The bounding of $\mathcal{H}_{\mathcal{P}\left(\mathcal{M}_{\text{adapt}}\right) \to \mathcal{P}(\mathcal{M})}$ ensures on the other hand, that $\mathcal{M}_{\text{adapt}}$ does not cover additional pixels on the screen. Other error measures like the quadric error metric [7] do not allow such guarantees. Figure 1 illustrates the mentioned cases for the projection of a 2D scene onto a 1D screen. The fine original mesh is draw in dotted style and the coarse, adapted mesh in bold style. On the 1D screen both projections are shown. The case when the coarse mesh does not cover all pixels covered by the fine mesh arises in area *1*. In area *2* both meshes project to the same pixels. The other erroneous case when the coarse mesh covers pixels that are not covered by the fine mesh is shown in area *3*. The grey arrows illustrate the one-sided Hausdorff distances – $\mathcal{H}_{\mathcal{P}(\mathcal{M}) \to \mathcal{P}\left(\mathcal{M}_{\text{adapt}}\right)}$ in area *1* and $\mathcal{H}_{\mathcal{P}\left(\mathcal{M}_{\text{adapt}}\right) \to \mathcal{P}(\mathcal{M})}$ in area *3*. The arrows originate from the most distant points on either mesh and point to the closest location on the other mesh.

Most hierarchies over irregular meshes are built from fine to coarse during the simplification of the irregular mesh with atomic simplification operations, such as vertex removal, edge or triangle collapse. The hierarchical model stores these atomic

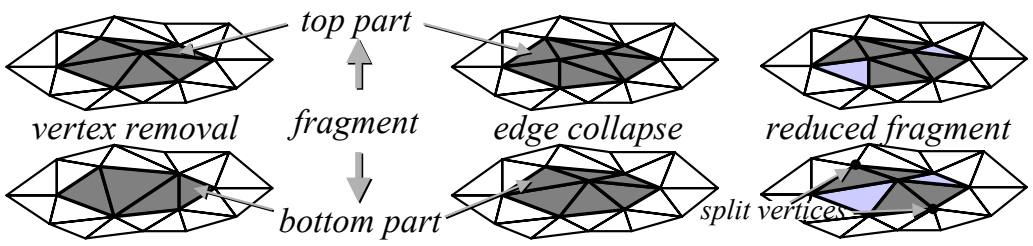


Figure 2. Illustration of the faces in the top and bottom part of a fragment for vertex removal on the left and edge collapse in the middle. On the right, reduced fragment used in view-dependent polygonal meshes.

coarsening operations, their inverse refinement operations (vertex insertion, vertex split) and the dependencies among them. The dependencies restrict the order, in which the refinement and coarsening operations can be applied during adaptive refinement. To balance the hierarchy the simplification algorithm is typically modified to select independent sets of operations. The balancing comes with a decrease in the quality of the coarsest approximations. Therefore the independent set based simplification can be steered by a rapidly computable error measure like quadric error metrics as well.

Two kinds of approaches have been used to define the dependencies among the different operations. The first kind [14,18] is based on the idea of the multi-triangulation (MT) [5]. Here the notion of a *fragment* is important, which is illustrated in Figure 2 for vertex removal and edge collapse simplification. For each coarsening operation a fragment is defined from two sets of faces: the *top part*, which is composed of all the faces that are changed by the coarsening operation, and the *bottom part* of the faces that replace the top part. The dependencies among the operations are defined therewith relative to the current tessellation: a coarsening/refinement operation is allowed iff all faces of the top/bottom fragment are present in the current tessellation. These dependencies ensure that all faces of all adaptively refined meshes have been present during simplification. The attributes stored within the hierarchy bound the different error terms conservatively for each face and typically over estimate the error.

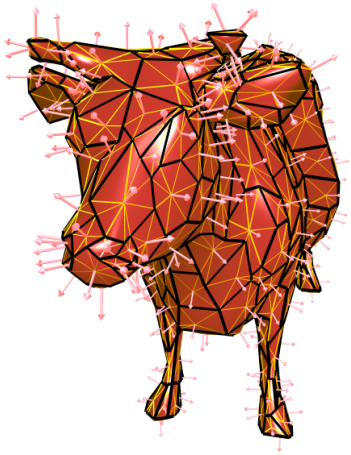
The major problem of MTs is the large number of dependencies among the atomic operations, which prohibit fast changes in the resolution and cost more storage space. The second kind of approaches like the view-dependent progressive meshes of Hoppe [9,16,6] therefore reduce the fragment size. Hoppe for example proposes to use only six triangles in the top and four in the bottom part as shown on the right of Figure 2. The improved adaptivity makes error control much more difficult. The affected faces of an edge collapse, which are not part of the fragment (lightly shaded faces in Figure 2), can assume different shapes depending on which of the edge collapse operations in their direct neighborhood is performed. To control the error, all of the different edge collapse configurations would have to be considered. As this is too expensive, this problem is either neglected [9] or more conservative

estimates [16,6] are used. The conservative estimates of Pajarola can only bound the one-sided Hausdorff distance and therefore cannot guarantee a correct image.

The work of Kim and Lee on truly selective refinement [11,12] aimed on the abolition of all dependencies among edge collapse and vertex split operations, that are not included in the vertex tree induced during edge collapse simplification. To accomplish this they identified each vertex of the original mesh with a dual patch around it in such a way that the whole surface is partitioned into dual patches. Each edge collapse simply unifies the dual patches of the collapsed vertices and assigns the union to the target vertex. In this way do the dual patches of any cut through the vertex tree partition the original surfaces. The two split vertices (compare Figure 2 on the right) necessary to perform a vertex split operation can be found by tracing the meeting points of the dual patches on the surface through the hierarchy. There are several problems with this approach. Firstly, there are additional test necessary to keep the connectivity consistent. But the major drawback is that the geometric error can be even less controlled as with view-dependent progressive meshes. The reason is that all faces in the one-ring of a vertex in a coarse approximation depend heavily on the edge collapses performed in the close neighborhood and a huge number of differently refined neighborhoods would have to be analyzed to bound the error – especially for the two-sided Hausdorff distance.

In this work a new mesh hierarchy is proposed based on the edge collapse operation performed on the dual mesh, which merges two faces. To equip the mesh hierarchy with coarse approximations, the merging of faces is interpreted on the primal domain as edge-removal operations. Vertices are eliminated whenever their valence decreases to two by edge-join operations as proposed in [8] for a progressive representation. Instead of a vertex tree the new hierarchy is built on a face tree or rather face forest. The hierarchy allows truly selective refinement of the face forest with control of the two-sided Hausdorff distance during selective refinement. No consistency checks for the connectivity are necessary and therefore no such checks restrict the selectivity as in [11].

The remainder of the paper is structured as follows. The next section reviews mesh simplification based on edge-removal and edge-join operations. Section 3 describes how the mesh hierarchy is built and what dependencies are stored. The error control is introduced in section 4 together with a proof of correctness. The paper is concluded with some examples.

a) 

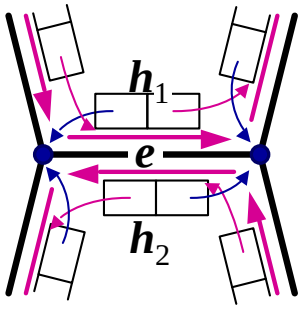
b) 

Figure 3. a)Illustration of the surface definition of a polygonal mesh with non planar faces. These are tessellated in a star (yellow lines) around the face center with the shown vertex normal.
b) the twinned half-edge data structure stores with each half-edge the origin vertex and the next pointer. The half-edges are sorted, such that the twinned half-edges are in successive order in the half-edge list.

2 Preliminaries

2.1 Polygonal Mesh Surface

The coarsening operations of edge-removal and edge-join as used in [8] can be applied to triangular as well as polygonal meshes and they always construct polygonal meshes as output. Although their order is optimized for planarity, the vertices of a face in the simplified model do not have to be coplanar. Therefore a useful definition of the surface within each polygon is necessary. I follow the ideas of [8] and tessellate each non triangular polygon with a star around a newly introduced face center vertex. The face center location $\boldsymbol{v_f}$ is computed as the centroid of the face vertices. A normal vector $\boldsymbol{n_f}$ is added to the face center in the direction orthogonal to a plane fitted to the face vertices. Figure 3 a) illustrates the polygonal mesh surface and visualizes the star shaped tessellation (yellow lines) and the added face center normals. For the rendering of the adaptive polygonal mesh it is useful to store the center locations and normals explicitly for all polygonal faces arising during simplification.

2.2 Twinned Half-Edge Data Structure

The connectivity of the adaptive polygonal mesh is stored in the space efficient twinned half-edge representation used in OpenMesh [2] and also described in Gum-

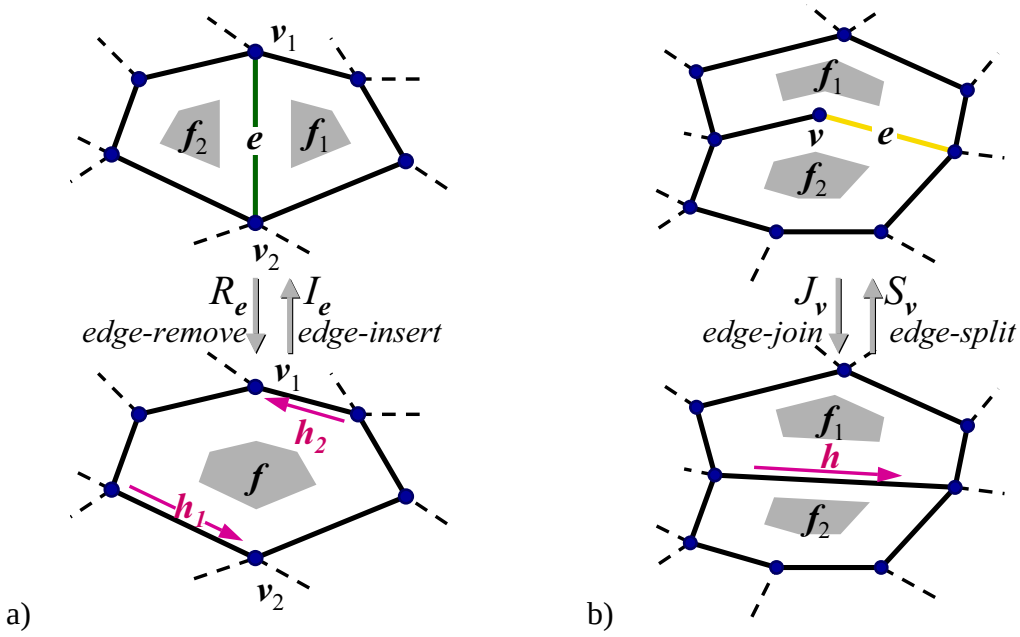


Figure 4. Basic simplification and refinement operations:
a) edge-removal merges faces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ into $\boldsymbol{f}$ / edge-insert is based on anchor half-edges $\boldsymbol{h}_1$, $\boldsymbol{h}_2$ and splits $\boldsymbol{f}$ into $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$.
b) edge-join eliminates $\boldsymbol{v}$ and merges adjacent edges / edge-split introduces vertex $\boldsymbol{v}$ by splitting the anchor half-edge $\boldsymbol{h}$.

hold [8]. It is a minimal half-edge data structure [17,10] with a special ordering of the half-edges. Figure 3 b) illustrates the two fields stored with each half-edge: the origin vertex and the next half-edge in the same face. The two half-edges composing an edge are stored in successive order in the list of half-edges such that the twin or inverse half-edge can be efficiently computed with a binary XOR 1 on the half-edge index. From the special order follows that also the edges can be used as basic elements of the data structure. The group of two twinned half-edges form a winged-edge [1] without the previous pointers. One can efficiently transform half-edge indices into edge-indices and vice versa by dividing/multiplying with two.

2.3 Polygonal Progressive Meshes

The proposed polygonal mesh hierarchy is built from a polygonal progressive mesh representation, which has been introduced in [8]. The simplification algorithm uses edge-removal and edge-join operations as illustrated in Figure 4. Each edge-removal operation $R_{\boldsymbol{e}}$ merges the two edge-adjacent faces into one face and decreases the valences of the two vertices incident upon the removed edge. The edge-join operations are performed automatically, whenever a vertex $\boldsymbol{v}$ of valence two arises to eliminate it with the join operation $J_{\boldsymbol{v}}$. In the dual mesh the join operation corresponds to the removal of a face of valence two. This is the dual-equivalent to the removal of the degenerate faces that arise during edge-collapse simplification. In

the pure triangular case every edge-collapse degenerates two triangles, that are automatically removed. In the polygonal progressive mesh representation zero, one or two edge-join operations can be induced by an edge-removal operation. Therefore, the edge-join operations are treated and stored separately.

The simplification algorithm used to build the polygonal progressive mesh sorts all edge-removal operations, into a priority queue. The induced edge-join operations also have to be performed before the evaluation of the error measure. Before an edge-removal is performed two consistency checks are performed and the operation is prohibited if one of them fails. The first check ensures that two faces never touch twice and that no face is adjacent to itself. The second check warrants that the star shaped tessellation around the face center stays a valid tessellation for all faces. The later test is similar to the normal flip test of edge-collapse based simplification.

The polygonal progressive mesh representation is finally composed of the base model (coarsest approximation) and a sequence of the inverse refinement operations. The inverse of the edge-removal is the edge-insert $I_{\boldsymbol{e}}$ operation as shown in Figure 4 a) from bottom to top. In the notation introduced for multi-triangulations consists the fragment of the edge-insert operation of only one polygonal bottom face $\boldsymbol{f}$, which is replaced by the two top faces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$. The polygonal mesh is not at all influenced outside of face $\boldsymbol{f}$.

The inverse of the edge-join is the edge-split $S_{\boldsymbol{v}}$ operation (Figure 4 b) bottom to top). It introduces the vertex $\boldsymbol{v}$. No new faces are created, such that both bottom and top part of its fragment consist of the faces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$.

Both of the atomic refinement and coarsening operations can be implemented very efficiently with the twinned half-edge data structure allowing for very fast changes in the resolution of the progressive polygonal mesh. In this work a rapid adaptive refinement framework is developed based on the edge-insert and edge-split operations.

3 Construction of Mesh Hierarchy

The polygonal mesh hierarchy is constructed from the progressive representation in a very similar way as view-dependent progressive meshes [9]. The progressive representation is read from disk and during the progressive reconstruction of the original model, the hierarchical representation is created as described in section 3.3. The main part of the hierarchy is a forest of binary trees over the faces, where each node corresponds to one polygonal face in the progressive model.

To balance the face hierarchy the original simplification algorithm [8] has been extended by an independent set selection technique as described in section 3.1.

As face cluster hierarchies typically have worse approximation qualities as vertex clustering hierarchies, I optimized the hierarchy with the variational shape approximation techniques of Cohen-Steiner et al. [4] as described in section 3.2

3.1 Independent Set Simplification

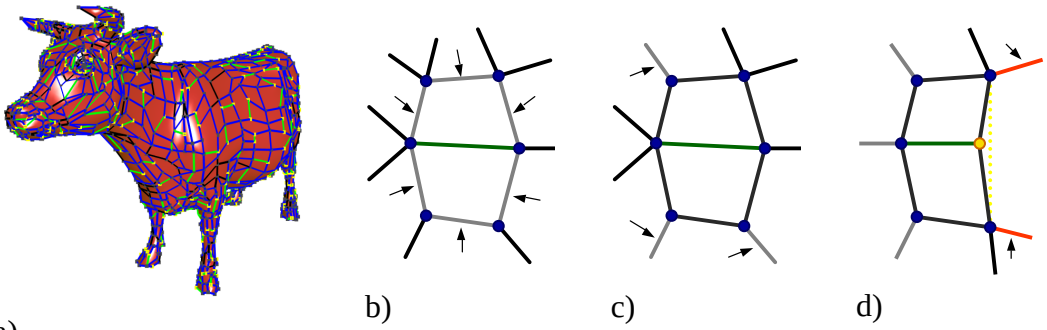


Figure 5. a) illustration of independent set selection with green edges to be removed, blue edges blocked, black edges invalid and yellow vertices to be removed. b/c) two kinds of to be blocked edges (grey) for the to be removed edge (green), d) to be updated and possibly re-queued edges highlighted in red.

In order to balance the face hierarchy I followed the well-known technique of independent set selection. The selection of independent sets is just a minor modification of the priority based simplification algorithm. The same priority queue is used to select edge-removal operations. Also the validity checks for connectivity and geometry are the same. The difference is that whenever an edge-removal operation from the top of the queue passes the validity checks, the operation is not directly performed, but only marked on the mesh. An edge- and a vertex-flag is allocated in order to mark the removed edges and vertices that are eliminated by edge-removal and edge-join operations. A second edge-flag is used to mark these edges as blocked, which would violate the independence of the selected set of operations. Figure 5 a) illustrates the marking with removed edges colored green, blocked edges blue and removed vertices yellow.

Figure 5 b) and c) illustrate the two kinds of edges that have to be blocked: b) all edges of the two edge-adjacent faces, c) all outer edges of valence three vertices that would cause an edge-join operation in the target face and change the error weight of the currently performed operation. Newly blocked edges are directly removed from the queue.

Different to most other independent set approaches is that edges are also re-inserted into the queue. These are the edges that had been invalid because of one of the geometric constraints and that could become valid after an edge-removal operation. d) shows these edges in red. The geometry of one of their adjacent faces is changed by the removal of the yellow vertex.

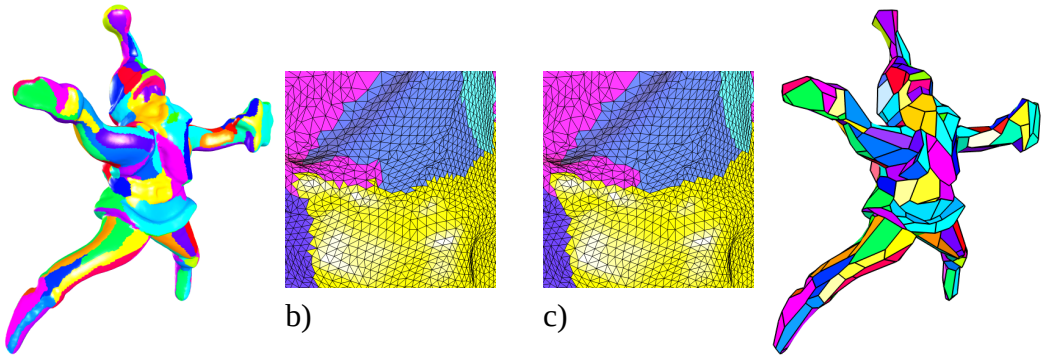


Figure 6. Hierarchy optimization with variational shape approximation: a) santa mesh clustered according to $\mathcal{L}^{2.1}$ norm into 200 clusters, b) close-up of zigzag boundaries between clusters, c) straightened boundaries, d) coarse model simplified with independent set face clustering

After the edges have been marked, all the coarsening operations are performed together and written to the progressive representation. The priority queue is re-built from the remaining edges and the next independent set is selected until only invalid edges remain.

3.2 Hierarchy Optimization with Variational Shape Approximation

As the approximation quality of polygonal progressive meshes is lower than edge-collapse based progressive meshes, I optimized the coarse resolution of the hierarchy with the variational shape approximation technique proposed by Cohen-Steiner et al. [4]. For a given number n of target clusters the $\mathcal{L}^{2.1}$ norm was used to cluster the faces into n face cluster as shown in Figure 6 a) with $n = 200$.

Then I used the proposed independent set based simplification algorithm but avoided the removal of all edges that are adjacent to two faces of different clusters. Minor problems arose at the cluster boundaries, which typically have a zigzag shape as shown in the close-up in Figure 6 b). Two simple cleanup procedures, which were applied between clustering and simplification, solved the problem. The objective of both cleanup procedures was the shortening of the cluster boundaries. The first cleanup moved single triangles to the dominant cluster in its neighborhood, i.e. if two or three edge-adjacent triangles belong to one different cluster the triangle itself is also added to this cluster. The second cleanup moved adjacent triangle pairs of one cluster to the dominant cluster in the neighborhood, if this reduced the length of the affected cluster boundary. Both cleanup proceedures where repeated until no more cluster changes happend. Typically, three to ten iterations were necessary. Figure 6 c) shows a close-up of the cleaned clustering in b).

Figure 6 d) shows the resulting simplified model. I examined several models and

target numbers of clusters. It turned out that a simplification to $1.5n$ faces was best. The approximations had significant smaller two-sided Hausdorff distance as compared with QSlim approximations of the same number of vertices. In terms of average squared error QSlim was superior in all cases, which is no drawback to the proposed method as it is based on controlling the Hausdorff distance. I decided to build the hierarchies with a target number of $n = 200$ clusters and turned off the cluster constraint during simplification after $2n$ faces had been reached, such that a complete hierarchy could be constructed.

The proposed combination of face clustering and variation shape approximation also simplifies the extremely complicated meshing approach described in [4], where it is hard to show that for an arbitrary clustering a coarse mesh can be found. The proposed approach on the other hand always generates a coarse polygonal mesh independent of the connectedness of the clusters. In the example of Figure 6 d) all faces in the meshed clustering allow for the star shaped tessellation around the face center. In order to generate convex faces I also implemented a consistency check for the simplification algorithm that ensures the convexity of all faces. In this way the clusters could also be meshed with convex faces.

3.3 Hierarchical Structure and Dependencies

Figure 7 gives an example of a polygonal mesh hierarchy. In a) the base model with two faces and nine vertices is shown. b) to d) show three levels that were generated by independent set removal of edges. e) is the original mesh and in f) an example for an adaptively refined mesh is depicted. Upper case letters label the faces in the hierarchy, lower case the vertices and Greek letters the edges. Inserted edges are labeled with "$\boldsymbol{e}_A$", where the subscript corresponds to the face that is split. In Figures a)-e) only the labels of newly introduced elements are shown, and all the unlabeled elements keep the name from the previous figures.

3.3.1 Face-Forest

The primary hierarchy of the polygonal multi-resolution model is built from the edge-removal operations that generate a forest of binary trees over the faces, as shown in Figure 7 g). The roots of the face-forest are the faces in the base mesh – in the example A and B – and the leaves are the faces of the original mesh, which can be triangular or polygonal. Each edge-removal/edge-insert corresponds to an internal node of the face-forest. In Figure 7 g) the removed/inserted edge is written underneath the node for the first two hierarchy levels. The parent and child relations of the tree are inherited onto the faces. For example is face A the parent face of C and D, and Q and R are the children faces of L.

Figure 8 illustrates the face-forest of the santa model. a) shows the base model with

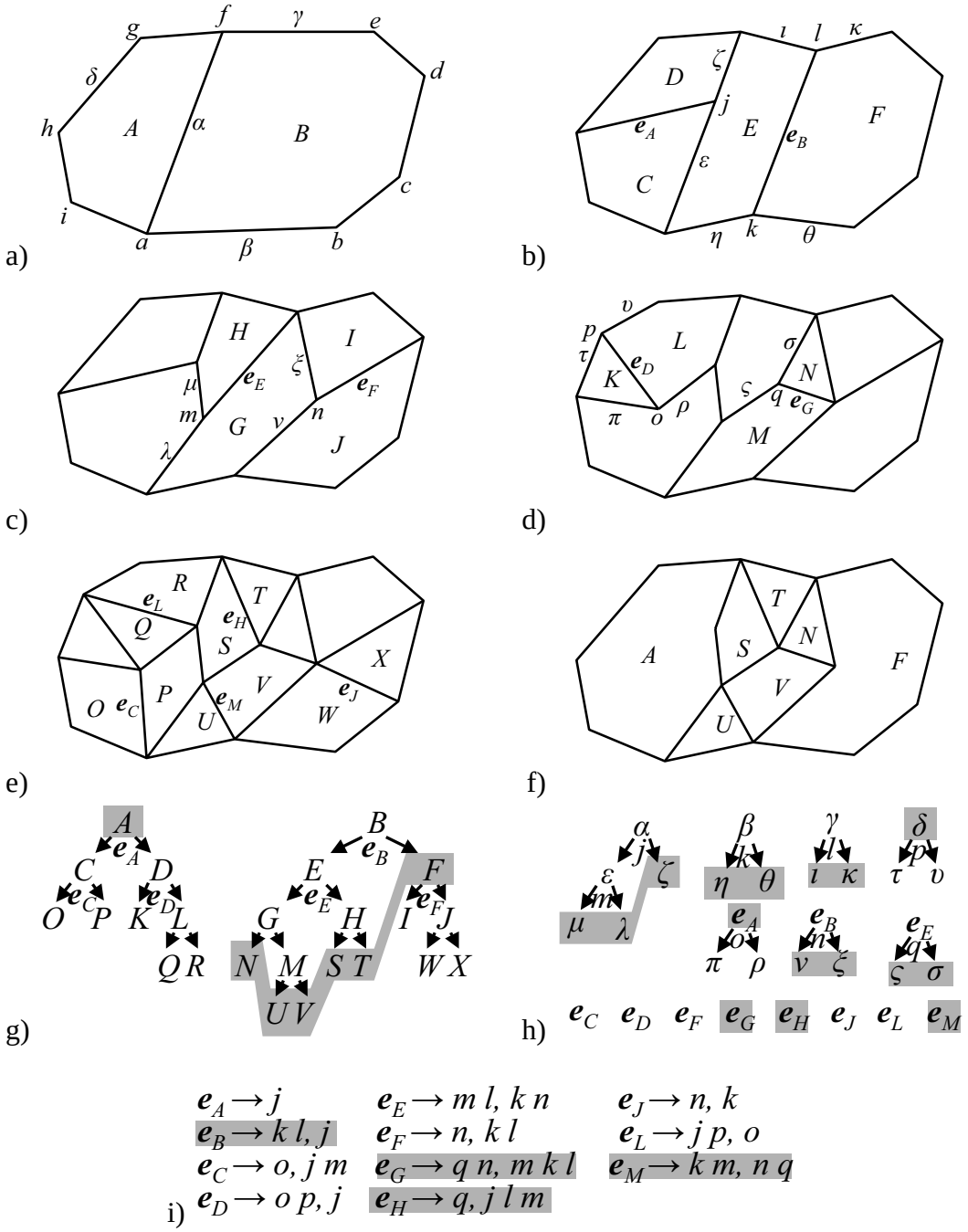


Figure 7. a-e) base model and four sets of refinements, upper case letters label faces, lower case vertices, and Greek letters edges; f) adaptively refined mesh, where the cuts through the face- and edge-forests and the face-vertex dependencies are shaded in g) face-forest, h) edge-forest and i) face-vertex dependencies.

the root faces. b) illustrates the patches of faces from the original surface that are merged by edge-removals into the same face of the base model. In c) and d) each face of the base model has been subdivided once, and in e) and f) two levels further down in the hierarchy are shown.

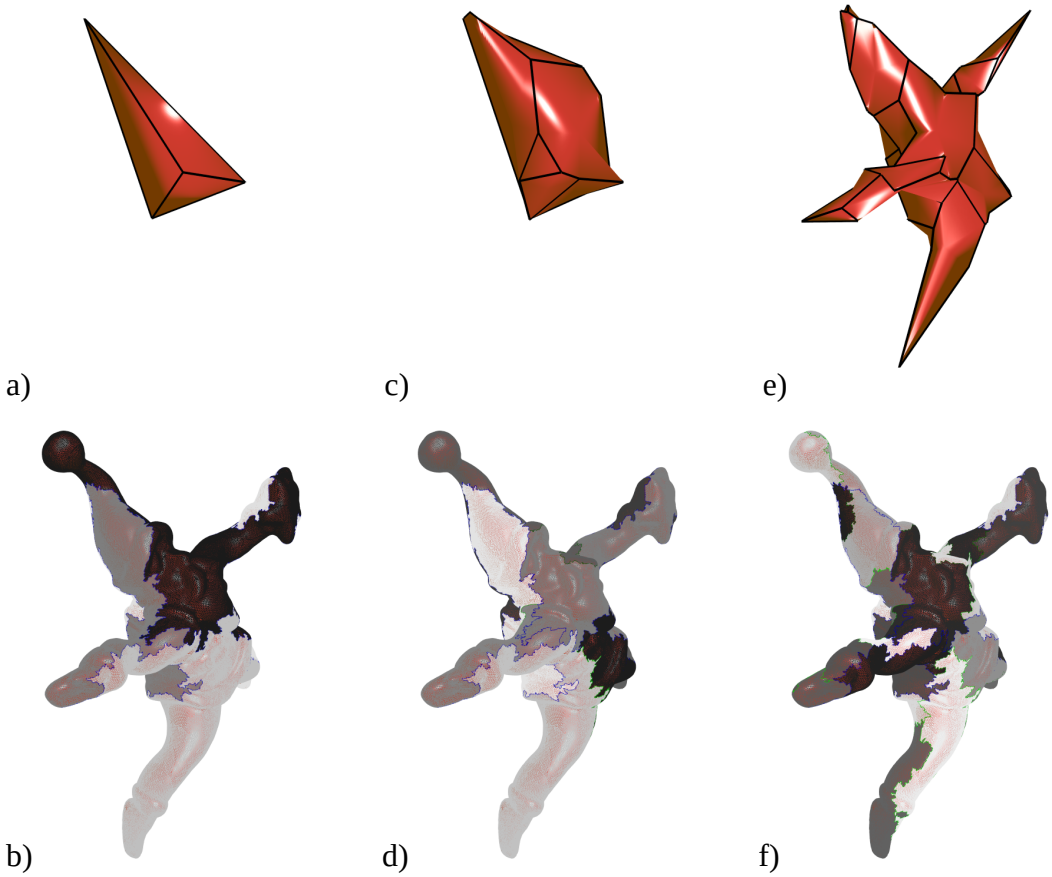


Figure 8. Illustration of the first levels of the face forest in the coarse representation and on the original santa mesh: a/b) base model: roots of forest, c/d) all faces refined once, e/f) third level of tree.

3.3.2 Edge-Forest

The edge-join / edge-split operations remove / introduce the vertices of the polygonal mesh. Figure 4 b) shows that each edge-split operation replaces one edge of the mesh by two edges and a new vertex. Although the space of the edge containing ***h*** before the split-operation is re-used for the left of the two new edges on top of Figure 4 b), one of the end vertices of this edge changes, such that both edges can be interpreted as new edges. In Figure 7 new Greek labels are given to both edges. The edges of successive split operations form a binary tree – an edge-tree. Over each edge of the base model and each edge introduced by an edge-insert operation a potentially empty binary edge-tree is formed. In Figure 7 h) the edge-forest of the hierarchy in a)-e) is shown. To each node corresponds the insertion/removal of the vertex written below the node. In Figure 8 b) all leaves of the edge-trees over the base edges are colored blue. d) and f) additionally show in green the leaves of the edge-trees over the insert-edges.

3.3.3 Dependencies

Let us illuminate the dependencies among edge-remove/insert and edge-join/split operations from the point of view of Multi-Triangulations. The bottom and top parts of the corresponding fragments are shown in Figure 4. Each edge-insert operation replaces the bottom face $\boldsymbol{f}$ with two top faces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$. This implies that all operations that need one of the faces $\boldsymbol{f}_{1/2}$ in the current tessellation depend on this edge-insert operation. Operations that are based on the bottom $\boldsymbol{f}$ depend on the edge-removal operation in 4 a). Similarly, depend operations that need the vertex $\boldsymbol{v}$ in Figure 4 b) on this edge-split operation. Therefore, it is sufficient to declare the dependencies of an operation by enumeration of all dependent faces and vertices.

In the setting of Multi-Triangulations the edge-insert operation in Figure 4 a) would thus depend on face $\boldsymbol{f}$ and all its vertices. The edge-remove would depend on $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ with all their vertices. The edge-split and edge-join operations in Figure 4 b) would only be allowed if both faces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are present with all their vertices.

The dependencies of the edge-split and edge-join operations restrict the selectivity of the hierarchical mesh noticeably. For truly selective refinement the only dependencies among the edge-remove/insert operations should be imposed by the face-forest (compare Figure 4 a): edge-remove is allowed, whenever faces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are not split; edge-insert is allowed, whenever face $\boldsymbol{f}$ is present in the current tessellation. These dependencies are called *face-face dependencies*. A valid selection of faces would then correspond to a cut through the face-forest as the one shaded in Figure 7 g), which corresponds to the adaptive mesh of f).

In the adaptive polygonal mesh the edge-join/split operations are performed automatically similarly to the simplification process. This automation is triggered by two further types of dependencies: the *face-vertex* and *vertex-vertex dependencies*. The vertex-vertex dependencies implement the dependencies among the edge-join/split operations that are induced by the edge-forest, i.e. for splits the parent edge has to be present and for joins the child edges must be present and not split.

The remaining face-vertex dependencies as shown in Figure 7 i) ensure that the vertices necessary for an edge-insert operation are present. The minimum requirement for the edge-insert in Figure 4 a) would be the presence of vertices $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. The minimum required vertices are listed in Figure 7 i) before the comma. The edge-join operations $J_{\boldsymbol{v}}$ on the other hand are triggered, whenever the vertex valence of vertex $\boldsymbol{v}$ becomes two. No explicitly stored dependencies are necessary for that.

To be able to efficiently control the two-sided Hausdorff distance one could go back to the idea of the Multi-Triangulation and introduce also vertex-face dependencies that ensure that both faces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ in Figure 4 b) are present before performing an edge-split operation. To keep the mesh hierarchy truly selective I instead increase the number of face-vertex dependencies and demand that whenever a face $\boldsymbol{f}$ is ac-

tive, at least all these vertices must be present that were part of the face during simplification. All these face-vertex dependencies are shown in Figure 7 i). The second advantage of the extended face-vertex dependencies is that they automatically ensure a valid connectivity.

To efficiently store the face-vertex dependencies the face-face dependency is used as precondition. Before the edge insertion of $\boldsymbol{e}_A$ between Figure 7 a) and b) the face-face dependencies tell us that face A with all its vertices is present in the current tessellation. Therefore only the vertex j has to be ensured by an explicitly stored dependency. Similarly is the insertion of $\boldsymbol{e}_E$ only dependent on face E and vertex m.

In order to efficiently store the face-vertex dependencies note that in Figure 4 b) only the two edge-insert operations that split faces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ can depend on the edge-split operation. Similar to the looping of fragments in [13] can one link all the edge-split operations, onto which a fragment depends, in a singly linked list. For this the link to the first edge-split operation is stored in the edge-insert operation and recursively the link to next edge-split within the referenced edge-split. Each edge-split needs to store two links, where each link is composed of an edge-split index and a bit, telling which of the two links of the referenced edge-split carries on the list. By packing each link into a 32-bit integer, only one integer per edge-insert and two per edge-split are necessary to store the insert-split dependencies.

All the introduced dependencies can be easily built while reading the progressive polygonal mesh representation with the help of two additional integer attributes. To construct the face-forest within each half-edge the index of the parent edge-insert operation is stored. For the construction of the edge-forest within each edge a link to the parent refinement operation is stored, where this time the link-bit tells if the parent is an edge-insert or an edge-split.

The proposed mesh hierarchy is truly selective in the sense that an arbitrary cut through the face forest can be selected without causing inconsistencies in the connectivity and with the ability of error control. Figure 7 f) shows an adaptively refined mesh of the hierarchy in a)-e). The corresponding cuts through the face- and edge forests are shown in g) and h).

3.4 Data Structure for Polygonal Multi-Resolution Mesh

The proposed polygonal multi-resolution model consists of the base model in the twinned half-edge data structure plus two lists of refinement operations: a list of edge-insert operations and a list of edge-split operations. The basic types of the data structure are 32-bit integers, 32-bit floats, 3d points and vectors with 32-bit float coordinates and *links*. A link is composed of an index and a flag, which are packed into a 32-bit integer. The flag can be used to specify the type of refinement

operation or to select one of two links to the next face-vertex dependent edge-split operations.

The records for edge-insert and edge-split first store the information necessary to perform the refinement on the current mesh. As the operations can be performed in arbitrary order, no fixed half-edge indices exist. Therefore, each half-edge is identified with the operation that it introduces, such that the necessary anchor half-edges $\boldsymbol{h}_{1/2}$ and $\boldsymbol{h}$ in Figure 4 a) and b) are stored as links to the introducing operations. A hash-map is used to map operation indices to half-edges of the current tessellation.

The second information stored within each refinement operation are the dependencies. Each edge-insert stores children and parent edge-insert operations and a link to the first face-vertex dependent edge-split operation. The edge-split operations only need to know their parent edge-split and store the two necessary next links of the face-vertex dependencies as discussed in section 3.3.3.

Finally is the geometry of the introduced mesh element stored for each operation. The edge-insert operations store face center and face normal of face $\boldsymbol{f}$ and the edge-split operations position and normal of the introduced vertex $\boldsymbol{v}$.

To analyze the storage space consumed for the polygonal multi-resolution model let us use v as the number of vertices, e the number of edges and f the number of faces of the input model and assume that the size of the base mesh is negligible. Then v is also the number of edge-split operations and f the number of edge-insert operations. The total number of refinement operations is $e = v + f$, i.e. the Euler equation without Euler characteristic. The consumed storage space $\mathcal{S}$ is measured in bytes per vertex (bpv) or bytes per edge (bpe) and is separated into connectivity (anchors), hierarchy (dependencies) and geometry (locations, normals and error bounds). From the discussion above one gets:

$$\mathcal{S} = \mathcal{S}_\mathcal{C} + \mathcal{S}_\mathcal{H} + \mathcal{S}_\mathcal{G} = (8f + 4v) + (16f + 12v) + (24f + 24v) = 48f + 40v.$$

To compare with Hoppe [9] and Pajarola [16] three additional float attributes per edge insert operation are necessary yielding $60f + 40v$ bpv. For a triangular mesh with $f = 2v$ the total storage space is 160 bpv. This is much less than the 224 bpv reported by Hoppe and only a factor 1.4 more than the 106 bpv reported by Pajarola. But my approach consumes less storage space for non triangulated polygonal models.

For practical applications can the geometry cost be reduced further by compressing the normal vectors into two bytes, the attributes into one byte each and the face center location and normal can be computed on the fly. After this compression only $\mathcal{S}_{\mathcal{G},\text{compressed}} = 3f+14v$ bytes remain. Also the hierarchy can be compressed further by using compressed representations of binary trees as for example proposed by Munro and Raman [15]. Only the information theoretic lower bound of two bits per

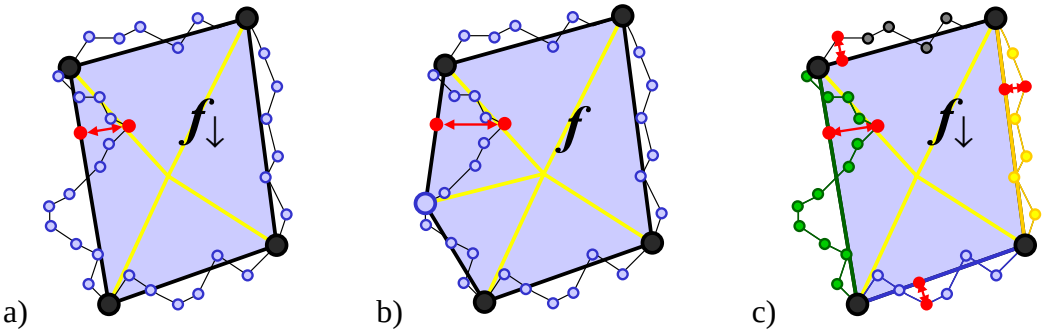


Figure 9. a) face $\boldsymbol{f}_{\downarrow}$ as generated by the simplification algorithm shown together with the boundary of $F(\boldsymbol{f}_{\downarrow})$, b) insertion of one vertex causes a larger Hausdorff distance $\mathcal{H}_{\boldsymbol{f}\leftrightarrow F(\boldsymbol{f})}$, c) illustration of parametric Hausdorff distance between coarse and fine boundaries

node are necessary with constant time access to parents and children. Therefore, the hierarchy can be compressed down to $\mathcal{S}_{\mathcal{H},\text{compressed}} = 4\frac{1}{2}f + 8\frac{1}{2}v$ bytes. The total storage space would reduce to $\mathcal{S}_{\text{compressed}} = 15\frac{1}{2}f + 26\frac{1}{2}v$ and for triangular models $57\frac{1}{2}$ bytes per vertex. In comparison does the twinned half-edge data structure with only two 32-bit pointers per half-edge and the same normal vector compression already consume 26 bytes per vertex, i.e. nearly half of the space necessary for a compact in-core hierarchical polygonal mesh.

4 Error Control

The two-sided Hausdorff distance fulfills the following partitioning property:

$$\mathcal{H}_{\mathcal{S}_1 \dot\cup \mathcal{S}_2 \leftrightarrow \mathcal{S}_a \dot\cup \mathcal{S}_b} \leq \max\left\{\mathcal{H}_{\mathcal{S}_1 \leftrightarrow \mathcal{S}_a}, \mathcal{H}_{\mathcal{S}_2 \leftrightarrow \mathcal{S}_b}\right\},$$

where the dot over the union sign implies $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$. In my hierarchical mesh representation any cut through the face-forest leads to an adaptive approximation $\mathcal{M}_{\text{adapt}}$ of the original mesh $\mathcal{M}$. The faces of $\mathcal{M}_{\text{adapt}}$ are a valid partition of $\mathcal{M}_{\text{adapt}}$ and each of the coarse faces $\boldsymbol{f}$ can be naturally identified with the set of faces $F(\boldsymbol{f})$ of $\mathcal{M}$ that are merged into $\boldsymbol{f}$. The sets $F(\boldsymbol{f})$ on the other hand partition $\mathcal{M}$ and the partitioning property of the Hausdorff distance allows to bound $\mathcal{H}_{\mathcal{M}_{\text{adapt}}\leftrightarrow\mathcal{M}}$ by bounding all $\mathcal{H}_{\boldsymbol{f}\leftrightarrow F(\boldsymbol{f})}$.

From the face-vertex dependencies follows that any face $\boldsymbol{f}$ of $\mathcal{M}_{\text{adapt}}$ at least contains the vertices, which were also incident on the instance $\boldsymbol{f}_{\downarrow}$ of $\boldsymbol{f}$ generated during simplification. But the face $\boldsymbol{f}$ of $\mathcal{M}_{\text{adapt}}$ can include further vertices. Figure 9 a) and b) show a simple example where the insertion of additional vertices increases the two-sided Hausdorff distance between $\boldsymbol{f}$ and $F(\boldsymbol{f})$. The spot of maximal distance is shown in red. For an efficient mesh hierarchy it is unavoidable to pre-compute error bounds. But for this one would have to consider all possible vertex insertions into a face $\boldsymbol{f}_{\downarrow}$ and measure the Hausdorff distance between all possible $\boldsymbol{f}$ and

$F(\boldsymbol{f}) = F(\boldsymbol{f}_\downarrow)$. This is clearly not feasible.

Instead I propose to bound the two-sided Hausdorff distance from above. This bound is the sum of the Hausdorff distance between $\boldsymbol{f}_\downarrow$ and $F(\boldsymbol{f}_\downarrow)$ and a parametric form of the Hausdorff distance between the boundary of $\boldsymbol{f}_\downarrow$ and the boundary of $F(\boldsymbol{f}_\downarrow)$. The boundary of a face or surface patch is denoted by the ∂-symbol. Figure 9 c) illustrates the proposed parametric Hausdorff distance $\tilde{\mathcal{H}}_{\partial \boldsymbol{f}_\downarrow \leftrightarrow \partial F(\boldsymbol{f}_\downarrow)}$ between the coarse and fine boundaries. For a boundary edge $\boldsymbol{e}$ of $\boldsymbol{f}_\downarrow$ let $E(\boldsymbol{e})$ define all the corresponding edges of the original mesh (in Figure 9 c) illustrated by different colors). Then the parametric Hausdorff distance of the boundaries is defined as

$$\tilde{\mathcal{H}}_{\partial \boldsymbol{f}_\downarrow \leftrightarrow \partial F(\boldsymbol{f}_\downarrow)} \stackrel{\text{def}}{=} \max_{\boldsymbol{e} \in \partial \boldsymbol{f}_\downarrow} \mathcal{H}_{\boldsymbol{e} \leftrightarrow E(\boldsymbol{e})}. \tag{3}$$

Similar definitions can be given for the one-sided parametric Hausdorff distances between coarse and fine boundaries. With this definition the theorem on the conservative error bound can be stated:

Theorem 1 (Conservative Error Bound) *Let $\boldsymbol{f}_\downarrow$ be a coarse face generated during simplification with boundary $\partial \boldsymbol{f}_\downarrow$ and $F(\boldsymbol{f}_\downarrow)$ the corresponding patch on the original mesh with boundary $\partial F(\boldsymbol{f}_\downarrow)$. Then holds for any $\boldsymbol{f}$ corresponding to $\boldsymbol{f}_\downarrow$ in an adaptively refined mesh*

$$\mathcal{H}_{\boldsymbol{f} \leftrightarrow F(\boldsymbol{f})} \leq \Xi(\boldsymbol{f}_\downarrow) \stackrel{\text{def}}{=} \mathcal{H}_{\boldsymbol{f}_\downarrow \leftrightarrow F(\boldsymbol{f}_\downarrow)} + \tilde{\mathcal{H}}_{\partial \boldsymbol{f}_\downarrow \leftrightarrow \partial F(\boldsymbol{f}_\downarrow))}. \tag{4}$$

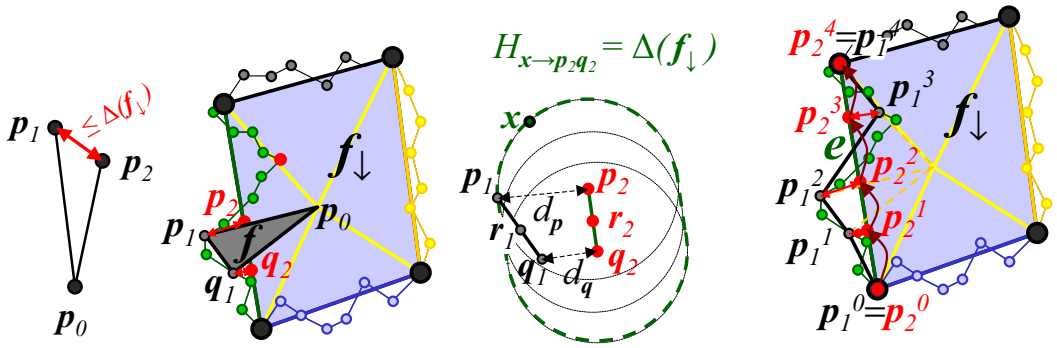
Thus the error bound $\Xi(\boldsymbol{f}_\downarrow)$ does only depend on the coarse face $\boldsymbol{f}_\downarrow$ generated during mesh simplification and its corresponding patch $F(\boldsymbol{f}_\downarrow)$ on the original surface. And it bounds for any refinement of the boundary of $\boldsymbol{f}_\downarrow$ the two-sided Hausdorff distance to $F(\boldsymbol{f}_\downarrow)$. In the mesh hierarchy for each edge-insert operation that splits a face $\boldsymbol{f}_\downarrow$ the bound $\Xi(\boldsymbol{f}_\downarrow)$ is stored and compared to the user defined error bound during adaptive refinement. During hierarchy construction I computed the Hausdorff distance between the faces and their corresponding patches on the original mesh with the Metro tool [3] and the parametric Hausdorff distance between the boundaries with a brute force method. In order to speed up the calls to Metro a ram-disk was used and the simple cases, where the boundary of $\boldsymbol{f}_\downarrow$ coincided with $F(\boldsymbol{f}_\downarrow)$, were handled explicitly.

Proof of Theorem 1: The first step in the proof is the application of the triangle inequality for one-sided Hausdorff distances to the distances between $\boldsymbol{f}$ and $F(\boldsymbol{f}_\downarrow)$:

$$\mathcal{H}_{\boldsymbol{f} \rightarrow F(\boldsymbol{f})} \leq \mathcal{H}_{\boldsymbol{f} \rightarrow \boldsymbol{f}_\downarrow} + \mathcal{H}_{\boldsymbol{f}_\downarrow \rightarrow F(\boldsymbol{f}) = F(\boldsymbol{f}_\downarrow)} \quad \text{and} \quad \mathcal{H}_{F(\boldsymbol{f}) \rightarrow \boldsymbol{f}} \leq \mathcal{H}_{F(\boldsymbol{f}_\downarrow) \rightarrow \boldsymbol{f}_\downarrow} + \mathcal{H}_{\boldsymbol{f}_\downarrow \rightarrow \boldsymbol{f}}$$

By comparing with inequality 4 it remains to prove

$$\mathcal{H}_{\boldsymbol{f}_\downarrow \leftrightarrow \boldsymbol{f}} \leq \Delta(\boldsymbol{f}_\downarrow) \stackrel{\text{def}}{=} \tilde{\mathcal{H}}_{\partial \boldsymbol{f}_\downarrow \leftrightarrow \partial F(\boldsymbol{f}_\downarrow))}, \tag{5}$$



a) b) c) d)

Figure 10. a) two meeting segments, b) for each inserted vertex on $\boldsymbol{f}$ exists a closest point (red) on the boundary of $\boldsymbol{f}_\downarrow$, c) for each $\mathbf{r}_1$ exists $\mathbf{r}_2$ with distance smaller than $\Delta(\boldsymbol{f}_\downarrow)$, d) connecting the projections $\mathbf{p}_2^i$ of the vertex locations $\mathbf{p}_1^i$ of $\boldsymbol{f}$ always covers the edge $\boldsymbol{e}$.

i.e. one has to show that the two-sided Hausdorff distance between the coarse face and the face with refined boundary is bound by the two-sided parametric Hausdorff distance between the boundary of the coarse face and the boundary of the corresponding patch on the original surface.

As the face center is stored within the mesh hierarchy, both faces $\boldsymbol{f}_\downarrow$ and $\boldsymbol{f}$ are star-shaped tessellations around the same center vertex $\mathbf{p}_0$. This allows to reduce the problem to segments from $\mathbf{p}_0$ to points on the boundary of $\boldsymbol{f}_\downarrow$ and $\boldsymbol{f}$. In the following points with subscript 1 will be on the boundary of $\boldsymbol{f}$ and with subscript 2 on the boundary of $\boldsymbol{f}_\downarrow$. Figure 10 a) shows such a constellation. Independent of lengths and angle between the segments can the two-sided Hausdorff distance be bound by the distance between the end-points of the segments:

$$\mathcal{H}_{\mathbf{p}_0\mathbf{p}_1 \leftrightarrow \mathbf{p}_0\mathbf{p}_2} \leq \|\mathbf{p}_1 - \mathbf{p}_2\| \,. \tag{6}$$

After this result remains the task to find for every point $\mathbf{p}_1$ on $\partial\boldsymbol{f}$ a point $\mathbf{p}_2$ on $\partial\boldsymbol{f}_\downarrow$ within a distance of $\Delta(\boldsymbol{f}_\downarrow)$ and vice versa.

For the first direction $\mathcal{H}_{\boldsymbol{f}\to\boldsymbol{f}_\downarrow} \leq \Delta(\boldsymbol{f}_\downarrow)$ one has to find for each point $\mathbf{p}_1$ on the boundary of $\boldsymbol{f}$ a point $\mathbf{p}_2$ on $\partial\boldsymbol{f}_\downarrow$ within a distance of $\Delta(\boldsymbol{f}_\downarrow)$. The mesh vertices on $\partial\boldsymbol{f}$ do also lay on the completely refined patch $F(\boldsymbol{f}) = F(\boldsymbol{f}_\downarrow)$ and the bound $\Delta(\boldsymbol{f}_\downarrow) = \tilde{\mathcal{H}}_{\partial\boldsymbol{f}_\downarrow \leftrightarrow \partial F(\boldsymbol{f}_\downarrow)}$ tells us that there exist points on $\partial\boldsymbol{f}_\downarrow$ no further apart than $\Delta(\boldsymbol{f}_\downarrow)$. Let $\mathbf{p}_1$ and $\mathbf{q}_1$ be two successive mesh vertices on $\partial\boldsymbol{f}$ and $\mathbf{p}_2$ and $\mathbf{q}_2$ the corresponding closest points on $\partial\boldsymbol{f}_\downarrow$ (compare Figure 10 b). From the properties of the parametric Hausdorff distance follows that one can choose $\mathbf{p}_2$ and $\mathbf{q}_2$ on the same boundary edge of $\boldsymbol{f}_\downarrow$, such that all points on the segment $\mathbf{p}_2\mathbf{q}_2$ are also on $\partial\boldsymbol{f}_\downarrow$. This leads us to the situation in Figure 10 c). Both distances $d_\mathbf{p}$ and $d_\mathbf{q}$ are $\leq \Delta(\boldsymbol{f}_\downarrow)$. The dashed green line around segment $\mathbf{p}_2\mathbf{q}_2$ is the boundary of the region of points $\mathbf{x}$ with one point on $\mathbf{p}_2\mathbf{q}_2$ within the range of $\Delta(\boldsymbol{f}_\downarrow)$, i.e. $\mathcal{H}_{\mathbf{x}\to\mathbf{p}_2\mathbf{q}_2} \leq \Delta(\boldsymbol{f}_\downarrow)$. As both points $\mathbf{p}_1$ and $\mathbf{q}_1$ are inside of this region and as the region is convex, all points $\mathbf{r}_1$ on the segment $\mathbf{p}_1\mathbf{q}_1$ are also inside the region. From $\mathcal{H}_{\mathbf{r}_1\to\mathbf{p}_2\mathbf{q}_2} \leq \Delta(\boldsymbol{f}_\downarrow)$ follows

the existence of a point $\mathbf{r}_2$ with $\|\mathbf{r}_2 - \mathbf{r}_1\| \leq \Delta(\boldsymbol{f}_\downarrow)$. This completes together with inequality 6 the proof of the first direction $\mathcal{H}_{\boldsymbol{f}\to\boldsymbol{f}_\downarrow} \leq \Delta(\boldsymbol{f}_\downarrow)$ of 5.

For the proof of $\mathcal{H}_{\boldsymbol{f}_\downarrow\to\boldsymbol{f}} \leq \Delta(\boldsymbol{f}_\downarrow)$ one has to find for each point $\mathbf{p}_2$ on the boundary of $\boldsymbol{f}_\downarrow$ a point on the boundary of $\boldsymbol{f}$, which is not further apart than $\Delta(\boldsymbol{f}_\downarrow)$. For this one projects the locations $\mathbf{p}_1^i$ of the mesh vertices of $\boldsymbol{f}$ onto their closest points $\mathbf{p}_2^i$ on the corresponding edge $\boldsymbol{e}$ of the boundary of $\boldsymbol{f}_\downarrow$ as depicted in Figure 10 d). From the bound on the parametric Hausdorff distance follows that this can be done such that $\|\mathbf{p}_2^i - \mathbf{p}_1^i\| \leq \Delta(\boldsymbol{f}_\downarrow)$ holds for all i. Thus the task has been solved for the points $\mathbf{p}_2^i$. The next step is to form the segments $\mathbf{p}_2^0\mathbf{p}_2^1$, $\mathbf{p}_2^1\mathbf{p}_2^2$, ..., which cover the edge $\boldsymbol{e}$ completely (with potential overlaps). With the same argument as illustrated in Figure 10 c) one can find for any point $\mathbf{r}_2$ on all the segments $\mathbf{p}_2^i\mathbf{p}_2^{i+1}$ a corresponding point $\mathbf{r}_1$ on $\mathbf{p}_1^i\mathbf{p}_1^{i+1}$ within the bound $\Delta(\boldsymbol{f}_\downarrow)$. As the segments cover $\boldsymbol{e}$, this holds for any point on $\boldsymbol{e}$, what completes the proof.

5 Results

Figure 11 demonstrates the selectivity of the proposed method at the isis model, where one of the four faces of the base model was completely refined. A large number of vertices is inserted on the boundary of the coarse faces, but the model is still consistent and the resolution varies maximally. The number of triangles necessary to tesselate the three coarse faces of the base tetrahedron depends on the length of the boundary, which is in the order of the square root of the faces in the refined patch.

Figures 11 b) to f) show examples of view-dependently refined meshes. The current rendering system allows for backface culling, view-frustum culling and screen space error control. For backface and view-frustum culling a normal cone around the face normal and a bounding sphere around the face center is computed for every face $\boldsymbol{f}$ in the hierarchy and stored with the edge-insert operation that splits this face. With these attributes the view-dependent refinement tests were implemented as proposed by Pajarol [16]. Figure 11 e) includes a side view with the view frustum and the unseen faces drawn as wireframe.

Table 1 tabulates measured quantities of the used models that are shown in the different Figures of the paper. The first two rows give the total number of vertices and edges of the models. The second two the corresponding numbers of the base model. The number of edge-split operations is simply $v - v_{\text{base}}$ and the number of edge-insert operations $e - e_{\text{base}}$. The large number of base faces for the croco model are due to the 65 unconnected components that all reduce to an isolated tetrahedron. The feline and buddha model are topologically more complex and have therefore larger base meshes. Finally, the blade model is topologically complex and has a large number of components.

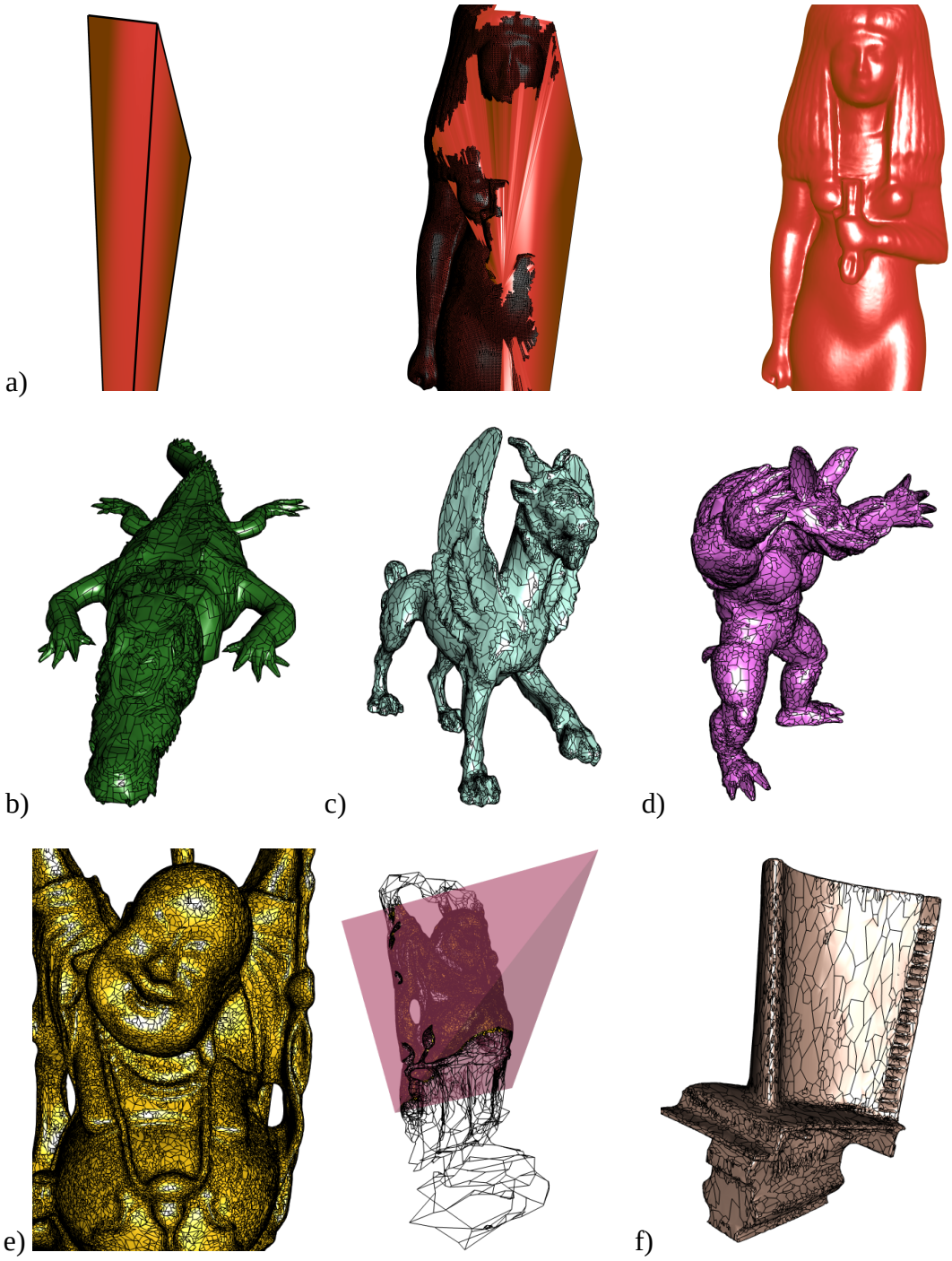


Figure 11. a) maximal refinement of one face of the base mesh of isis model, b)-f) view-dependently refined meshes for a screen space error of one pixel at a resolution of 400x570 pixels with backface and view-frustum culling, and error control, e) includes a side view of the adapted model, where the faces that are not seen on the screen were drawn empty

model	cow	croco	feline	santa	isis	armadillo	buddha	blade
v	2904	17332	49864	75781	187644	172974	543652	882954
e	8706	51606	149598	227337	562926	518916	1631574	2648082
v_{base}	4	260	57	4	4	4	2817	4294
e_{base}	6	390	86	6	6	6	4780	6596
d	27	32	48	48	58	58	55	87
d_{IS}	18	25	29	27	30	30	29	34
t_{cluster}	1.3	8.7	18.4	40.2	130.2	112.1	2012.1	589.1
t_{simp}	0.7	4.1	12.4	19.3	51.3	48.5	1531.7	262.4
t_{hier}	0.8	0.2	0.4	0.8	2.0	2.1	7.0	14.0
t_{spheres}	0.0	1.6	0.8	0.6	1.6	1.5	246.5	680.6
t_{cones}	0.2	2.8	4.5	5.8	14.9	14.2	283.6	758.3
t_{error}	51	431	1464	1667	5202	5851	36713	99809
$\mathcal{S}$	431	2546	7401	11248	27853	25676	80656	130931

Table 1
Sizes and construction timings for the measured models. The rows tabulate: model name, total number of vertices and edges, vertices and edges in base model, maximum hierarchy depth without and with independent set approach, number of levels, clustering time, simplification time, construction times for hierarchy, bounding spheres, normal-cones, Hausdorff distance and storage space in KB (including base model).

Rows five and six compare the maximal depth of the hierarchy for the standard simplification algorithm and the independent set approach. It can be clearly seen that the independent set approach balances the hierarchies very well. The balanced hierarchies also result in much faster construction times. In the successive rows only the construction times for the independent set approach are given. All timings are written in seconds and were measured on a Pentium IV with 2.4 GHz. The table separates the running times for clustering, simplification, hierarchy construction, bounding sphere computation and Hausdorff-distance measurement. The hierarchy construction time includes reading the progressive representation from disk and is similarly to bounding sphere computation negligible. The normal-cone computation is still clearly faster than simplification. Although the Hausdorff-distance computation with the Metro Tool is the bottle neck at the moment, it allows for a lot of optimization and has running time of $O(n \log n)$. For the models with more than 500k vertices, there is a jump in the running-time, which is not in accordance to the $O(n \log n)$ complexity. The reason is that the current implementation allocates too much storage space and forces the operating system into memory swapping mode.

Table 2 tabulates the mesh element counts for the view-dependent renderings in

model	v_{adapt}	e_{adapt}	f_{adapt}	t_{adapt}
croco	9855	15262	4479	23599
feline	15693	22725	5777	36047
armadillo	20130	27537	6161	46195
buddha	107046	167334	51353	262942
blade	39813	53997	12217	92779

Table 2
View-dependent statistics for the adaptive meshes of Figure 11 b)-f): number of vertices, number of edges, number of faces and the number of rendered triangles in the adaptive mesh.

Figures 11 b)-f), which were all acquired at a screen resolution of 400x570 pixels with a screen space error of one pixel. The last column shows the actual number of the triangles, which were used to tessellate the visible faces. This number if slightly more than twice the number of vertices, which is the factor expected for a pure triangle mesh. The selective refinement of the polygonal mesh hierarchy is extremely fast. Measurements on the same Pentium IV 2.4 GHz show in average that one to two million edges can be inserted or removed per second.

6 Conclusion

A new mesh hierarchy was introduced that builds on edge-removal/insert and edge-join/split operations. It allows for the first time truly selective refinement with control of the two-sided Hausdorff distance. No initial triangulation of polygonal models is necessary. Furthermore, a new method has been proposed that optimizes the face cluster hierarchies with the help of variational shape approximation. This combination also yielded a simple solution to the meshing problem of the variational shape approximation approach.

In future work it is planed to tackle the problem of potential normal flips that can arise in the current solution as a cause of vertex insertions. It is probably possible to restrict the hierarchy construction in a way that no flips are possible. Another interesting direction for future work is to optimize the whole hierarchy, not only one target number of faces and to apply the proposed scheme in different applications.

References

[1] B. Baumgart. A polyhedron representation for computer vision. *Proceedings of AFIPS '75*, 44:589–596, June 1975.

[2] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Openmesh – a generic and efficient polygon mesh data structure. In *Proceedings of OpenSG Symposium*, 2002.

[3] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[4] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Transactions on Graphics*, 23(3), August 2004. Proc. SIGGRAPH 2004, To appear.

[5] L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multiresolution hypersurface modeling. In R. Rau W. Straßer, R. Klein, editor, *Geometric Modeling: Theory and Practice*. Springer-Verlag, 1997.

[6] C. DeCoro and R. Pajarola. XFastMesh: Fast view-dependent meshing from external memory. In *Proceedings of Visualization '02*, pages 363–370, 2002.

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

[8] S. Gumhold. Polygonal progressive meshes. Technical Report WSI–2004–4, Wilhelm-Schickard-Institut für Informatik, University of Tübingen, Germany, July 2004.

[9] H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of ACM SIGGRAPH '97*, pages 189–198, August 1997.

[10] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. In *Proceedings of ACM Symposium on Computational Geometry '98*, 1998.

[11] J. Kim and S. Lee. Truly selective refinement of progressive meshes. In *No description on Graphics interface 2001*, pages 101–110. Canadian Information Processing Society, 2001.

[12] J. Kim, S. Lee, and L. Kobbelt. View-dependent streaming of progressive meshes, 2004. to appear in Shape Modeling International 2004.

[13] R. Klein and S. Gumhold. Data compression of multiresolution surfaces. In *Visualization in Scientific Computing 98*, pages 13–24, April 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] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, June 2002.

[16] R. Pajarola. FastMesh: Efficient View-Dependent meshing. In B. Werner, editor, *Proceedings of Pacific Conference on Computer Graphics and Applications '01*, pages 22–30, October 2001.

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

[18] J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, 1997.