Volume xx (*200y*), *Number z, pp. 1–12*

Stellar Mesh Simplification Using Probabilistic Optimization

Antônio W. Vieira[1,2] , Thomas Lewiner[1,3], Luiz Velho[4], Hélio Lopes[1], and Geovan Tavares[1]

[1] PUC–Rio — Laboratório Matmídia — Rio de Janeiro
[2] UNIMONTES — CCET — Montes Claros
[3] INRIA — Géométrica Project — Sophia Antipolis
[4] IMPA — Laboratório Visgraf — Rio de Janeiro

Abstract

This paper introduces the Stellar Simplification *scheme, a fast implementation of the Four–Face Cluster algorithm. In this scheme, a probabilistic optimization heuristic substitutes the priority queue of the original algorithm, which results in a 40% faster algorithm with the same order of distortion. This implementation uses a very concise data structure which consists only of two arrays of integers to represent the surface topology. The method extends naturaly to a progressive and/or multi–resolution computational scheme for combinatorial surfaces.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

The typical surface models handled by contemporary Computer Graphics applications have millions of triangles. Mesh Simplification has emerged as a critical step for handling such huge meshes. On one hand, there is an evident need for removing redundancies on meshes obtained by surface reconstruction algorithms, such as Marching Cubes [LC87]. On the other hand, meshes with a high level of detail even without redundancies could be extremely expensive to render, to store or to transmit [Hop96]. In both cases, an efficient simplification process would generate simpler models with lower computational costs [CMS98]. This paper introduces the *Stellar Simplification* scheme, a fast implementation of the Four–Face Cluster algorithm. It uses a probabilistic optimization heuristic that substitutes the priority queue of the original algorithm. The result is a 40% faster algorithm with the same order of distortion. It also produces naturaly a hierarchical multi–resolution structure that can be used in a wide range of applications [PS97].

Prior work. Many efficient mesh simplification methods are based on local topological operators. The most common is the *Edge–Collapse*, which consists in contracting the two vertices of an edge onto a unique vertex, eliminating its two incident faces. The inverse of the Edge–Collapse operation is called *Vertex–Split*. Simplification schemes construct a sequence of edges to be collapsed, resulting in a hierarchy of meshes of decreasing size. Traversing this sequence in a reverse order, with *Vertex–Split* operations, it is possible to recover details from the mesh of minimal size (base mesh). Although topological tests can preserve the topology of the surface during the simplification, its geometry will be distorted. To minimize this distortion, an energy function measuring the quality of the mesh is used to guide the mesh simplification. An example of such function is the quadric error metric [GH97]. For those algorithms, the priority queue is a natural data structure to store the order of the edges to be simplified, since it allows a variety of operations (inclusion, removal of the largest, etc.) to be efficiently performed.

Although the priority queue is efficient, it is necessary to build it before starting the simplification process. Moreover, at each step of the process a time is consumed not only to reprocess the geometrical change for all edges involved in the local operation, but also to update their priority on the queue. In order to accelerate this process, Wu and Kobbelt [WK02] presented a technique called Multiple–Choice based on probabilistic optimization where there is

no use of a priority queue, but the edge to be contracted is chosen among a small number of randomly selected edges.

In [Vel01], Velho proposed a mesh simplification method, called Four–Face cluster (FFC) mesh simplification, which produces a sequence of *Edge–Weld* operations intercalated with *Edge–Flip* operations. Edge–Weld and Edge–Flip are two example of topological operators based on the Stellar Theory. The Edge–Flip swaps an internal edge of a 2 face cluster. The Edge–Weld removes a valence 4 vertex from a four–face cluster and replaces it by the internal edge of a two–face cluster. Edge–flips are required to change the valence of a vertex to 4. Stellar theory [Ale30, New26] proved that those two operators forms a complete set for changing the connectivity of the mesh without modifying its topology. Differently from the other algorithms cited above, the FFC computes the energy function on the vertices, and not on the edges.

Contributions. This work enhances the Four–Face Clusters algorithm by substituting the priority queue by the Multiple-Choice technique. The result is an algorithm that is about 40% faster than the original one and that requires less memory. We call the new algorithm *Fast Stellar Simplification*, because it is based on the stellar operators Edge–Weld and Edge–Flip. To provide a formal definition for those operators, an introduction to Stellar Theory is given.

We also introduce a new scheme to represent the hierarchy of meshes obtained during the process of simplification. In this scheme, we adopt the Corner–Table data structure [RSS01], which allows a concise and simple implementation of our new algorithm. Finally, we propose a practical way to compress the hierarchy of the multi–resolution mesh.

Paper outline. Sections 2 and 3 will review some basic concepts of topology and Stellar Thoery. Sections 4 and 5 describe the data structure and the topological operators that will be used in the algorithm. Section 6 presents the original Four–Face cluster algorithm. Section 7 introduces the Fast Stellar Simplification Algorithm based on the multiple choice technique. Section 8 presents a scheme to encode and decode the hierarchy of the surfaces generated by the simplification algorithm. Finally, section 9 shows some results and comparison with the original Four–Face cluster algorithm.

2. Surface Topology

A simplex σ^p of dimension p (p-simplex, for short) is the convex hull of $p+1$ points $\{v_0,\ldots,v_p\}\mathbb{R}^m$ in general position, i.e., the vectors $v_1-v_0, v_2-v_0,\ldots,v_p-v_0$ are linearly independent. The points $v_0,\ldots,v_p$ are called the vertices of σ. A face of σ is the convex span of some of the vertices of σ and therefore is also a simplex. The simplices of dimensions 2 and 1 will be called, respectively, triangles, edges. If σ is a face of a simplex τ then σ is said to be incident to τ. The boundary of a p-simplex σ, denoted by $\partial\sigma$, is the collection of all of its proper faces, i.e., different from σ itself. Two k-simplices σ and $\rho \in K$ are *adjacent* when $\sigma\cap\rho\neq\emptyset$. The valence or degree of a vertex $v \in K$ is the number of edges which have v as a vertex, and is denoted by $\deg(v)$.

A simplicial complex K is a finite set of simplices together with all its subsimplices such that if σ^p and τ^q, $p\leq q$, belong to K, then either σ^p and τ^p meet at a subsimplex λ^r, $r\leq p$, or are completely disjoint. In the last case, σ^p and τ^p are said to be *independent*.

The *underlying polyhedron* $|K|\subset\mathbb{R}^m$ corresponds to the union of the simplexes in K. A triangle mesh is the underlying polyhedron of a 2-dimensional simplicial complex.

The *join* $\sigma_1\star\sigma_2$ of independent simplices σ_1 and σ_2 is the simplex whose vertices are those of σ_1 and σ_2. The join of complexes K and L, written $K\star L$, is $\{\sigma\star\tau : \sigma\in K,\tau\in L\}$ if the following holds:

1. If $\sigma_1\in K$ and $\tau_1\in L$, σ_1 and τ_1 are independent.
2. If $\sigma_1,\sigma_2\in K$ and $\tau_1,\tau_2\in L$, then $\sigma_1\star\tau_1\cap\sigma_2\star\tau_2$ is either empty or a face of $\sigma_1\star\tau_1$ and $\sigma_2\star\tau_2$.

Consider a simplicial complex K and $\sigma\in K$. The local neighborhood of σ is basicaly described by the following definitions:

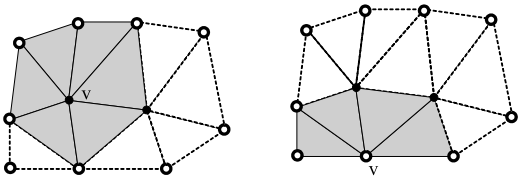
- The *open star* of σ is
$$star(\sigma,K)=\{\tau\in K : \sigma \text{ is a face of } \tau\}.$$
- The *star* of σ is
$$\overline{\text{star}}(\sigma,K)=\{\tau\in K : \tau \text{ is a face of an element of } star(\sigma,K)\}.$$
- The *link* of σ is
$$link(\sigma,K)=\{\tau\in K : \tau \text{ and } \sigma \text{ are independent and } \sigma\star\tau\in K\}.$$

Definition 1 (combinatorial surface) A triangular mesh M is a combinatorial surface if: Every edge in M is bounding either one or two triangles and the *link* of a vertex in M is homeomorphic either to an interval or to a circle.

The edges in a combinatorial surface M incident to only one face are called *boundary edges*. A vertex incident to a boundary edge is called a *boundary vertex*. The set of those boundary simplices forms the *boundary* of M and is denoted by ∂M. The boundary of a combinatorial surface is a collection of closed curves. The edges and vertices that are not on the boundary are called, respectively, *interior edges*[†] and *interior vertices*. Figures 1 and 2 illustrates the star and the link of interior and boundary vertices on a surface.

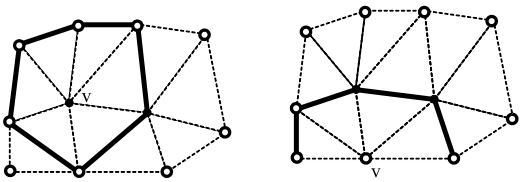
A combinatorial surface is *orientable* when it is possible to choose a coherent orientation on its edges, i.e., two adjacent triangles induce opposite orientations on their common edge.

[†] Notice that, on a combinatorial surface, the *Link* of an interior edge (u,v) are the opposite vertices by the edge (u,v).



(a) Interior vertex (b) Boundary vertex

Figure 1: *Vertex star.*



(a) Interior vertex (b) Boundary vertex

Figure 2: *Vertex link.*

From now on, a surface will allways mean an oriented combinatorial surface. This is the general case of manifold triangle meshes embedded in $\mathbb{R}^3$. For more details in the following definitions, see [Ede02].

3. Stellar Theory

In the previous section, we defined several concepts about topology that will be used in this work. Now, we will see how to manipulate the structure of a combinatorial surface *without* modifying its topology.

Stellar theory was developed in the 1920's, by [Ale30] and [New26]. It combines the abstract and piecewise linear approaches to combinatorial topology. More recently, [Pac91] consolidated the theory. The main point of Stellar theory is the study of equivalences between simplicial complexes [Lic99].

As we have seen in Section 2, the link and the star of a simplex σ provide a combinatorial description of the neighborhood of σ. We can use them to define certain changes in a triangle mesh, without modifying essentially (i.e., "topologically") that neighborhood. That is, we do not want to change the topology of the realization of the surface in $\mathbb{R}^3$.

The *stellar operations* provide a such change. They comprise *bistellar moves* and *stellar subdivision*.

Definition 2 Let K be an n-dimensional simplicial complex. Take an r-simplex $\sigma \in K$, and an $(n-r)$-simplex $\tau \notin K$, such that $link(\sigma, K) = \partial\tau$. Then, the operation $\kappa(\sigma, \tau)$, called *bistellar move*, consists of changing K by removing $\sigma \star \partial\tau$ and inserting $\partial\sigma \star \tau$.

The bistellar moves are atomic operations that make local changes to the neighborhood of an simplex, while maintaining the integrity of its combinatorial structure.

In the case of combinatorial surfaces, there are three types of bistellar moves, for $\dim\sigma = 2, 1, 0$, called 2-move, 1-move, and 0-move. They are shown in figure 3.

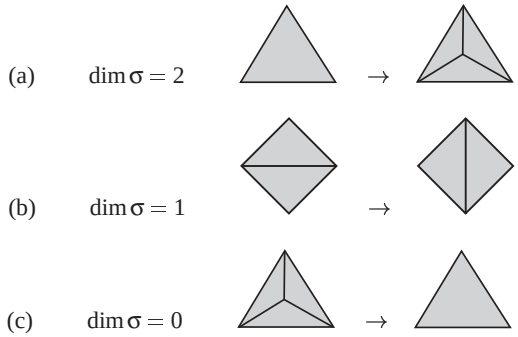


Figure 3: *Two dimensional bistellar moves.*

The fundamental result of the Stellar theory is given by the theorem below.

Theorem 3 ([New26], [Pac91]) Two combinatorial surfaces are piecewise linearly homeomorphic if and only if they are bistellar equivalent.

The above result guarantees that bistellar moves can change any triangulation of a closed piecewise linear manifold to any other.

A version of this theorem for manifolds with boundary uses all stellar operations, including stellar subdivision [Pac91].

Definition 4 Let K be a 2-dimensional simplicial complex, take an r-simplex $\sigma \in K$ and a vertex v in the interior of σ. The operation (σ, v) removes $\overline{star}(\sigma, K)$ and replaces it with $v \star \partial\sigma \star link(\sigma, K)$ is called a *stellar subdivision*.
The inverse operation $(\sigma, v)^{-1}$ is called a *stellar weld*.

Note that, some of the stellar subdivision and welds are also stellar moves. For example, in the two dimensional case, for $\dim\sigma = 2$, see $\kappa(\sigma, v)$ and $\kappa(v, \sigma)$ that are shown in the top and bottom rows of figure 3.

The new operation in two dimensions, is the stellar subdivision on edges, called 1-split. It is shown in figure 4 the interior edge case and in figure 5 the boundary edge case.

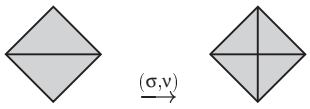


Figure 4: *Two dimensional stellar subdivision on interior edges.*

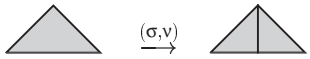


Figure 5: *Two dimensional stellar subdivision on boundary edges.*

Stellar subdivision is a very powerful concept and it is the cornerstone of Stellar theory. Here, we will only mention some results of the stellar subdivision theory [Ale30].

Proposition 5 Any stellar move, $\kappa(\sigma,\tau)$, is the composition of a stellar subdivision and a weld, namely $(\tau,v)^{-1}(\sigma,v)$.

This result can be easily seen through an example, shown in figure 6.

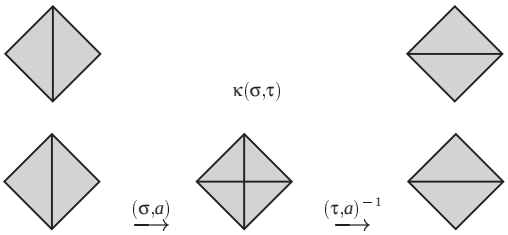


Figure 6: *A bistellar move on an edge can be decomposed into a subdivision and an weld.*

Proposition 6 Any stellar operation can be decomposed into a finite sequence of elementary stellar operations on *edges*.

This result is even stronger than the previous one. It basically allows us to restate the main theorem of Stellar theory only in terms of operations on edges.

4. Extended Corner–Table Data Structure

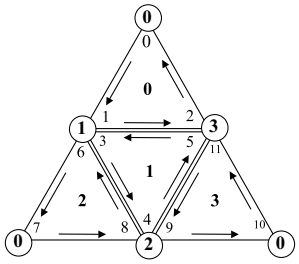
The Corner–Table (CT) is a very concise data structure for triangular meshes [RSS01]. It uses the concept of *corner* to represent the association of a triangle with one of its bounding vertices, or equivalently the association of a triangle with its opposite bounding edge to that corner: it may be viewed as a compact version of the half–edge representation for triangular meshes.

In this data structure, the corners, the vertices and the triangles are indexed by non–negative integers. Each triangle is represented by 3 consecutive corners that define its orientation. For example, corners 0, 1 and 2 correspond to the first triangle, the corners 3, 4 and 5 correspond to the second triangle and so on... As a consequence, a corner with index c is associated with the triangle of index $trig(c) = floor(c/3)$.

The Corner–Table data structure represents the geometry of a surface by the association of each corner c with its geometrical vertex index $V[c]$.

Assuming a counter–clockwise orientation, for each corner c, the $next(c)$ and $prev(c)$ corners on its triangle boundary are obtained by the use of the following expressions: $next(c) = 3 \times trig(c) + [(c+1) \bmod 3]$, and $prev(c) = 3 \times trig(c) + [(c+2) \bmod 3]$.

The edge–adjacency between the neighboring triangles is represented by associating to each corner c, its opposite corner $O[c]$, which has the same opposite edge. This information is stored in two integer arrays, called the V and O tables. Figure 7 shows an example of a Corner–Table representation for a tetrahedron.



V[0]	= 0	O[0]	= 4
V[1]	= 1	O[1]	= 9
V[2]	= 3	O[2]	= 8
V[3]	= 1	O[3]	= 10
V[4]	= 2	O[4]	= 0
V[5]	= 3	O[5]	= 7
V[6]	= 1	O[6]	= 11
V[7]	= 0	O[7]	= 5
V[8]	= 2	O[8]	= 2
V[9]	= 2	O[9]	= 1
V[10]	= 0	O[10]	= 3
V[11]	= 3	O[11]	= 6

Figure 7: *Tetrahedron example.*

The CT concisely represents the connectivety of a triangular mesh using only the arrays O and V. To represent the mesh geometry, an array G is used to store the geometry of the vertices (coordinates, normal,...).

Notice that, the number of entries of the O and V arrays is the number of corners on the mesh, i.e., three times the number of triangles. And the number of entries of the G array is the number of vertices on the mesh.

Where are the edges? The CT do not need an explicit representation for edges because they are implicitly represented by its opposite corners. A corner c also represents an edge whose endpoints are $V[prev(c)]$ and $V[next(c)]$.

To obtain the incidences and adjacencies of edges, two arrays CE and EC could be temporary allocated. The first array stores one corner $c = CE[e]$ that associates the edge e with one of its opposite corners c and the second array stores the edge $e = EC[c]$ that associate the corner c with its opposite edge e. This arrays will be used only to illustrate the incidences and adjacencies of an edge and can be allocated in linear time from the arrays O and V.

In figure 8 the edge a_1 is interior to the mesh and it could be represented by the corners a or d, while the edge a_2 is a boundary edge and can only be represented by the corner i.

It is important to notice that in our simplification algorithm we don't need to use the CE and EC arrays. But they can be very useful to develop an efficient implementation of other algorithms that are based on the edge–collapse operation.

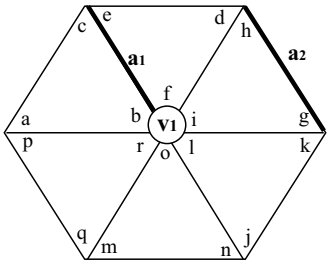


Figure 8: *Edge representation.*

Toward efficiency on vertex queries. In our new algorithm, a very important step is to obtain the vertex star efficiently, since it is based on edge–weld. Therefore, we extend the original CT by allocating an extra integer array VC that for each vertex v stores an index of a corner whose vertex is v. For example, in the figure 8 the vertex v_1 can be represented by the corner o. Only one corner is sufficient because all the incidence and adjacency relations can be obtained through the use of the O and V arrays according to the following algorithm.

Given a vertex v, we get the corner c associated to v and, in time O(*deg(v)*), obtain the list of adjacent vertices and incident edges and faces, denoted respectively by $v<V>, v<E>$ and $v<F>$

Algorithm 1: $\overline{\text{star}}(v)$

$c = j = VC[v]$;
$v<V>= \emptyset$; $v<E>= \emptyset$; $v<F>= \emptyset$;
Do
  $v<V>= v<V> + V[next(j)]$;
  $v<E>= v<E> + EC[next(j)]$
  $v<F>= v<F> + \lfloor j/3 \rfloor$;
  $j = next(right(j))$;
While $(j != c)$

5. Local Operators and Topological Validation

Our algorithm is based on two local topological operators: the *Edge–Flip*, and the *Edge–Weld*. In this section we will describe not only those two, but also the more classical *Edge–Collapse* operator in order to compare them.

Edge–Collapse: This operator consists in removing an edge $e = (u,v)$ of a surface S, identifying its vertices to a unique vertex $\overline{v}$. From a combinatorial viewpoint, this operator will remove 1 vertex, 3 edges and 2 faces from original mesh, thus preserving its Euler characteristic. From a geometric viewpoint, the new position of the vertex $\overline{v}$ can be computed with the geometry around u and v.

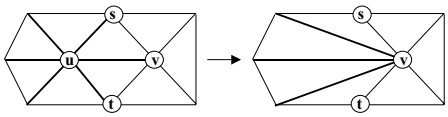


Figure 9: *Edge–Collapse.*

Let s and t be the vertices opposite to $e = (u,v)$, which is the edge to be collapsed (see figure 9). Choosing interior edges according to the following *collapse condition* will guarantee the topological consistency of this operation .

Lemma 7 (Collapse Condition) [HDD*93] Let S be a combinatorial 2–manifold. The collapse of an edge $e = (u,v) \in S$ preserves the topology of S if teh following conditions are satisfied:

1. $link(u) \cap link(v) = link(e)$;
2. If u and v are both boundary vertices, e is a boundary edge;
3. S has more than 4 vertices if neither u nor v are boundary vertices, or S has more than 3 vertices if either u or v are boundary vertices.

Figure 10(a) shows a valid operation and 10(b) an invalid. Figure 11 illustrates two examples of invalid collapse operations on boundary edge.

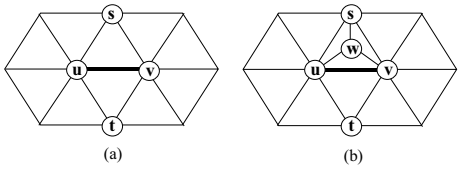


Figure 10: *Examples of a valid and an invalid collapse operation by the first condition.*

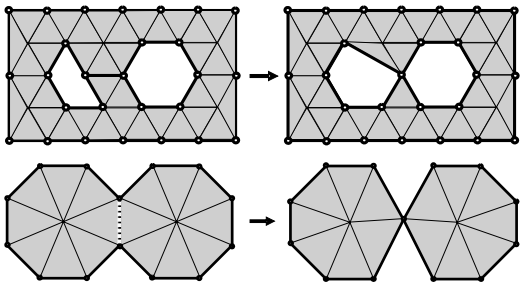


Figure 11: *Invalid edge-collapse operations by the second condition.*

Edge–Flip: This operation consists in transforming a two–face cluster into another two–face cluster by swapping its common edge.

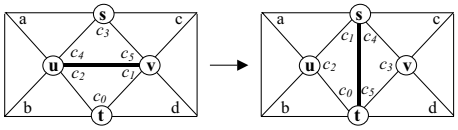


Figure 12: *Edge–Flip.*

Let consider the interior edge $e = (u,v)$ of a surface S and s and $t \in S$ the two vertices opposed to e (see figure 12). The Edge–Flip operation will replace e by (s,t), and replace the 2 triangles incident to e by (u,s,t) and (v,t,s). The condition to apply this operator is the following.

Lemma 8 (Flip Condition) [HDD*93] Let S be a combinatorial 2–manifold. The flip of an interior edge that replaces $e = (u,v) \in S$ by (s,t) preserves the topology of S if and only if $(s,t) \notin S$.

In the Corner–Table data structure, each edge can be univocally represented by one of its opposite corners. This is used by the Algorithm 2 to perform the *Edge–Flip* operation on the edge opposite to the corner c_0.

Algorithm 2: Edge–Flip(c_0)

// Label incident corners
$c_2 = prev(c_0)$; $c_1 = next(c_0)$;
$c_3 = O[c_0]$; $c_4 = next(c_3)$; $c_5 = prev(c_3)$;
$a = O[c_5]$; $b = O[c_2]$; $c = O[c_4]$; $d = O[c_1]$;
// Label incident vertices
$t = V[c_0]$; $v = V[c_1]$; $s = V[c_3]$;
// Perform swap
$V[c_1] = s$; $V[c_3] = v$; $V[c_4] = s$; $V[c_5] = t$;
// Reset opposite corners
$O[c_2] = c_3$; $O[c_3] = c_2$;
$O[c_0] = a$; $O[a] = c_0$;
$O[c_4] = d$; $O[d] = c_4$;
$O[c_5] = c$; $O[c] = c_5$;

Edge–Weld. This operation consists in transforming a four–face cluster into a two–face cluster by removing its central vertex.

Consider an interior valence 4–vertex v, adjacent to the vertices s, t, w and u (see figure 13). The star of v forms a four–face cluster. The Edge–Weld operation removes the vertex v and re–triangulates the quadrangle by splitting it. The dividing edge e can be either (w,u) or (s,t). The result is a two–face cluster composed by the two faces incident to e.

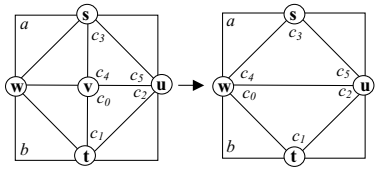


Figure 13: *Edge–Weld.*

Corollary 9 Consider a combinatorial 2–manifold M, and an interior vertex v of M with valence 4. With the notations of figure 13, the removal of the vertex v by the Edge–Weld operation preserves the topology of M if and only if there is no edge in M connecting w to u.

Proof: Consider s, w, t and u, in this order, the vertices adjacent to v. Removing v is equivalent to an Edge–Collapse operation on the edge $e = (v,u)$. Therefore, the topology of M is preserved if and only if the *collapse condition* is satisfied: $link(v) \cap link(u) = \{s,t\} = link(e)$. As $\{u,w\} = link(v) \setminus link(e)$, this condition is valid if and only if w does not belong to $link(u)$. This means that there is no edge connecting w to u. □

Given a mesh represented by a Corner–Table, the Algorithm 3 performs the removal of the vertex incident to the corner c_0:

Algorithm 3: Edge–Weld(c_0)

// Assign incidences
$c_1 = next(c_0)$; $c_2 = prev(c_0)$;
$c_4 = next(O[c_1])$; $c_5 = prev(O[c_1])$;
$a = O[next(O[c_5])]$; $b = O[prev(O[c_2])]$;
// Perform vertex removal
$V[c_0] = V[O[c_2]]$; $V[c_4] = V[c_0]$;
// Reset opposite corners
$O[c_5] = a$; $O[a] = c_5$;
$O[b] = c_2$; $O[c_2] = b$;

When an edge–weld is applied to remove a boundary vertex with valence 3, the edge-collapse condition should be observed.

As a consequence of proposition 5, one can prove that the *Edge–Collapse* operation can be decomposed into a sequence of *Edge–Flips* operations, followed by one *Edge–Weld* operation. Figure 14 shows a sequence example.

The combination of Edge–Weld and Edge–Flip stellar operations provides more flexibility than the edge–collapse operation itself.

6. The Four–Face Cluster Method

The Four–Face Cluster algorithm (FFC) is based on the Edge–Weld and Edge–Flip operators. Since the Edge–Col-

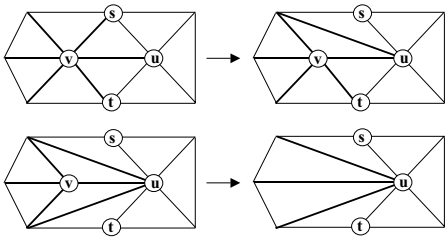


Figure 14: *Edge–Collapse decomposition*

lapse operation is equivalent to a sequence of those stellar operators, Edge–Collapse based method is a special case of the FFC method. However, stellar operations are more flexible. In the case of the Edge–Collapse / Vertex–Split, there are many possible sequences of Edge–Flips leading to a final Edge–Weld. Therefore, the order of those Edges–Flips can be chosen to improve the quality of the mesh (for example improving aspect ratio or preserving dihedral angle), which is more tricky in the Edge–Collapse operations. As a conclusion, the Four–Face Cluster (FFC) algorithm is more flexible than Edge–Collapse based algorithms.

Algorithm's outline. The FFC algorithm constructs a hierarchy of meshes $\left(M^0, M^1, \ldots, M^n\right)$ with a decreasing number of elements (vertices, edges and faces). In this hierarchy, surface M^0 is the original surface M, and each surface M^j, $j = 1 \ldots n$, corresponds to the level of detail j of M. We will now describe how we obtain such a hierarchy.

In the initialization step for each level of detail j, all the vertices of M^{j-1} are marked as valid for removal. Then, we select a marked vertex v to be removed from the surface M^{j-1}. When the valence of the vertex v is not 4, we apply a sequence of Edge–Flip operations to bring its valence to 4. The four vertices on the star of the modified vertex v are then unmarked. They will not be valid for further removal until the next level of detail $j+1$. Next, we remove the valence 4 vertex v using an Edge–Weld operation. Figure 14 illustrates this sequence of operations. When all the vertices are unmarked, the resulting surface M^j corresponds to level of detail j inside the hierarchy of surfaces. On entering the next step, we mark all the remaining vertices of M^j in order to create M^{j+1} and so on. The selection of the vertices to be removed is performed according to geometrical costs, such as the Quadratic Error Metric [GH97].

Quadric Error Metric. Each local simplification introduces a geometric error between the surfaces M^j and M^{j+1}. This cost is computed in the original FFC algorithm using the QEM [GH97], as follow.

Let v be a vertex of M^j, and $f_i \in \overline{\text{star}}(v)$ a face incident to v. Let a_i be the area of f_i and $p_i = (n_x, n_y, n_z, d)$ the plane supporting f_i. The *fundamental error quadric* Q_i is defined in [GH97] by:

$$Q_i = p_i p_i^T = \begin{pmatrix} n_x^2 & n_x n_y & n_x n_z & n_x d \\ n_x n_y & n_y^2 & n_y n_z & n_y d \\ n_x n_z & n_y n_z & n_z^2 & n_z d \\ n_x d & n_y d & n_z d & d^2 \end{pmatrix}$$

It can be used to compute the squared distance $d(w)$ of a point w to the plane p_i:

$$d_i(w) = w^T \left(p_i p_i^T\right) w = w^T Q_i w$$

Garland and Heckbert compute their quadric error metric by adding all of those distances $d_i(w)$ for all face f_i.

For boundary edge collapse, should exist criterias that preserve the geometrical aspect of the boundary. An alternative suggested by Garland, that uses QEM [GH97], constructs for each boundary edge a quadric that corresponding to the plane perpendicular to the boundary (see 15) and adds each quadric related to the extreme vertices of such edge. Using this strategy, the edges that contribute to the geometric aspect of the boundary are penalized with a high cost of contraction.

Figure 15: *Plane perpendicular to the boundary.*

Simplification criterion. The FFC algorithm assigns a quadric Q_v to each vertex v of the mesh, which is computed as the weighted sum of the quadrics Q_i associated with each face f_i incident to v:

$$Q_v = \sum a_i Q_i$$

We compute the error introduced by removing a vertex v from the mesh, considering each Edge–Flip and Edge–Weld costs, by [Vel01]:

$$E(v) = \alpha C(v) + \beta S(v)$$

This cost balances the vertex removal distortion $C(v)$ and the swap distortion $S(v)$. In our implementation, we chose $\alpha = 0.75$ and $\beta = 0.25$. The cost $C(v)$ of removing a vertex v of valence 4 is defined as:

$$C(v) = \min\left\{u^T (Q_v + Q_u)\, u, \;\; u \in link(v)\right\}$$

The cost $S(v)$ is the sum of the cost of Edge–Flips in $Star(v)$ used to bring v to valence 4. To compute this cost, we compute the sequence of independent Edge–Flips, minimizing the aspect ratio [Gue99] and trying to preserve the dihedral angle.

Implementation. The original implementation used a priority queue, which requires computing the costs of all vertices before beginning the simplification. Moreover, at each simplification step, we need to re–compute the cost of all vertices involved in the local operation. With a priority queue, we would have to update corresponding priorities. The Algorithm 4 gives the pseudo–code for the Four–Face Cluster Simplification Algorithm introduced in [Vel01]. It generates n levels of resolution of a given mesh M.

Algorithm 4: SimplifyFFC(M,n)

```
assign quadrics;
for all(v ∈ M) do
  compute E(v)
for (j = 1 to n) do
  mark all vertices as valid for removal
  insert all vertices in the priority queue
  while (queue is not empty) do
    get v from queue
    if (v marked) then
      perform edge swaps to bring v to valence 4
      unmark the vertices w ∈ link(v)
      remove vertex (v)
      re–compute the errors Q_u and Q_w
      update queue for w ∈ link(u) ∪ link(w)
```

7. The Fast Stellar Simplification Algorithm

An alternative to simplification methods based on priority queue was presented by Wu and Kobbelt in [WK02]. They propose, instead of using the priority queue, to choose the element to be simplified inside a reduced, randomly selected set of d elements. This probabilistic optimization strategy is motivated by the fact that when simplifying high resolution meshes, most of the vertices will be removed anyway. As a consequence, it would not be necessary to choose a vertex with lower cost among all possible candidates at each simplification step.

The basic idea of this Multiple–Choice technique is to obtain a small random subset of d edges to be collapsed, say $d = 8$, and perform the collapse for the edge with the lower cost among them. Our implementation is an adaptation of this idea, since we compute costs on vertices and not on edges. We obtain a random subset of vertices to be simplified and perform the simplification for the vertex with the lower cost of this set. This strategy leads to a good choice of the vertex to be simplified, except for a rare probability that all d random vertices were among the vertices that should not be removed. Assuming that we simplify a mesh down to 5% of its original complexity, we hope that the resulting (not removed) vertices will have high costs. The probability of choosing one of the high–cost vertices on a random subset of $d = 8$ vertices is actually small enough:

$$\left(\frac{5}{100}\right)^8 \approx 4 \times 10^{-11}$$

Using this technique we do not need to sort the vertices by their cost, avoiding the expansive use of a priority queue. Since, at each randomly selected subset the costs of the d vertices will be recomputed, it is also not necessary to re–compute the cost for the vertices on the neighborhood of the removed vertex. This allows a simpler implementation and improves the algorithm's performance. The Algorithm 5 shows the pseudo–code to generate n levels of resolution for a mesh M using this process:

Algorithm 5: SimplifyFS(M,n)

```
assign quadrics;
for (j = 1 to n) do
  mark vertices as valid for removal
  while (exist a valid vertex) do
    for (i = 1 to 8) do
      v_i = valid random vertex
      if E(v_i) < E(v) then v = v_i
    perform edge swaps to bring v to valence 4
    unmark the vertices w ∈ link(v)
    remove vertex (v)
    re–compute quadrics Q_u and Q_w
```

When the number of valid vertices is less then 8, one or more vertices can be selected more than once in order to remove all valid vertices.

8. Simplification with Multi–Resolution

One objective of this section is to introduce two strategies to encode the hierarchy of meshes $\left(M^0, M^1, .., M^n\right)$ generated by our Fast Stellar simplification algorithm: the *parallel* and the *sequential* encoding. The main difference between those two strategies relies in the type of hierarchical structure they produce. The parallel encoding produces a multi–resolution mesh with separated levels of details, whose resolution changes globally on each level. The sequential encoding produces a progressive mesh, whose resolution changes locally inside each level.

Multi–resolution representation. The Corner–Table data structure uses only two arrays (O and V) of integers to represent the surface topology and a third array (G) to store the geometry of the mesh. In order to represent the hierarchy of surfaces, in our implementation we adopt the following strategy. The connectivity of the surface M^j is represented by the arrays O^j and V^j, while its geometry uses the original array G for all levels of detail, since we do not modify the indices of the vertices during the simplification process. Therefore,

in order to change from one level to another, we simply allocate the arrays O and V corresponding to the desired level.

Parallel Encoding. This strategy produces a multi–resolution mesh whose resolution changes globally at each level. Parallel encoding is useful when the user wants to move from one level of resolution to another. In such application, we could allocate only the memory necessary to represent the surface of a certain level. Thus, we simply need to encode the surface M^j. We implemented this encoding in a compressed manner (with less than 2 bits per triangle) using the Edgebreaker compression scheme.

We used Edgebreaker because there is a very concise and simple implementation of this compression scheme for surfaces with handles using the Corner–Table data structure [LRS*02]. We modify this implementation by adding a fourth attribute to vertex geometry, its index on the G table of the original surface. Maintaining these indices for the vertices at each level of detail enable us to interchange the use of parallel and sequential encoding. For example, in the application we can move directly to the level 5 of resolution using the parallel encoding, and then in a progressive way return to the level 4 by the use of the sequential encoding.

Sequential Encoding. In the sequential case, we encode each stellar operation: at each vertex simplification we store the history of the Edge–Flips and Edge–Welds operations. This strategy is similar to the Progressive Mesh encoding [Hop96], which encodes the history of Edge–Collapse operations for their simplification algorithm. The main idea of this encoding is to obtain a refinement process by reading in the reverse order the history of stellar operations performed by the Fast Stellar simplification algorithm.

In the sequential encoding, we store a string of integers to represent the sequence of operations performed on each vertex simplification. Figure 16 illustrates a vertex simplification with the encoding of the corresponding operations.

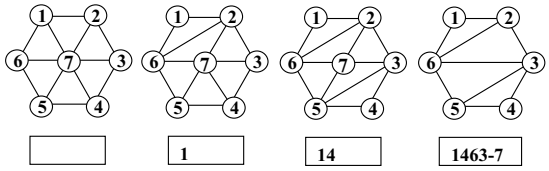


Figure 16: *Encoding of a vertex simplification.*

Let call v the vertex to be removed, and $e = (u, v) \in \overline{\text{star}}(v)$ an edge to be swapped. Since each e has v as extremity, we encode only the index of u to represent an Edge–Flip operation. The Edge–Weld operation is encoded by 3 integers 2 integers for encoding the resulting edge of the vertex removal and 1 integer to encode the original index of the removed vertex.

In order to interchange the parallel encoding with the sequential encoding, we write each level of detail in a separated file. As a consequence, we start the progressive refinement process at any level of detail obtained by the simplification. In the Progressive Mesh [Hop96], the starting surface for refinement is restricted to be the base mesh.

9. Results

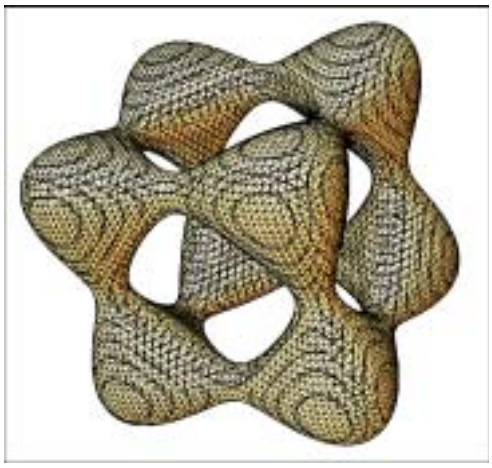
The results presented in [WK02] shows that the performances of mesh simplification algorithms using the Multiple–Choice technique are 2.5 times faster than using a priority queue, with similar outputs. Furthermore, the Multiple–Choice technique leads to a simpler implementation, since there is no need for a priority queue construction or of re–computation at each simplification step. Similarly, the experiments below show that the Fast Stellar simplification is 40% faster than the original Four–Face Cluster algorithm with the same order of distortion.

Figure 17 shows two intersting implicit surfaces examples obtained by the use of a Marching Cube algorithm [LLVT03]. Notice that the surface on those figures have the same topological type. One can visually observe that the geometry of both objects are very well preserved, including the sharp edges, altough the algorithm is performing a probabilistic optimization.

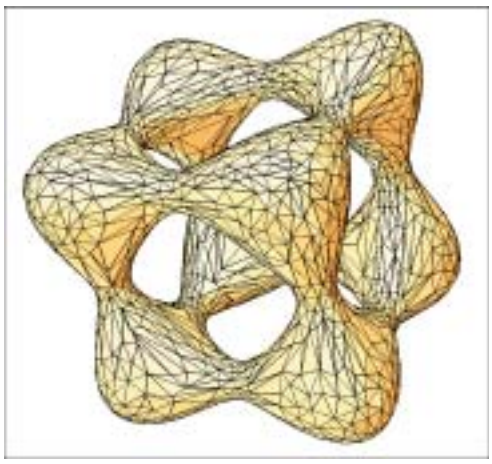
Table 1 shows the running time of each routine during the simplification, using the Four–Face Cluster (FFC) and Fast Stellar (FS) algorithms. Table 2 shows the total running time comparison (in *sec*) for simplifying some models. Figure 18 compares the absolute maximum geometric error, based on QEM, introduced for the *Bunny* model in various levels of details by the two different algorithms.

In figure 22 we present two surface models to visually compare the FFC and the FSS algorithms. Notice that, as in [WK02], the results got in both methods are similar. The first example is the *Bunny* model. Figure 23 shows the levels of detail $2, 4, 8$ and 11 obtained using the FFC and the FS. Figure 24 shows the simplification of *cow* model, our second example. We show in this figure the levels of detail $6, 8, 10$ and 12 obtained again using both algorithms.

In order to verify the influence of d in the algorithm, the number vertices used on the multiple-choice step, two graphics have been generated. For each d from 2 to 20, we run the algorithm in order to simplify the *Dino* original surface with 7400 triangles to a surface with 500 triangles (see figure 19). One graphic shows the total algorithm time as a function of d and the other shows the absolute maximum geometric error, based on QEM, again as a function of d. They are respectively illustrated in figures **??** and **??**. As a conclusion, setting d as 8 is really a nice decision, since the total time varies linearly with d and there is no substancial gain on error reduction for d greater than 8.

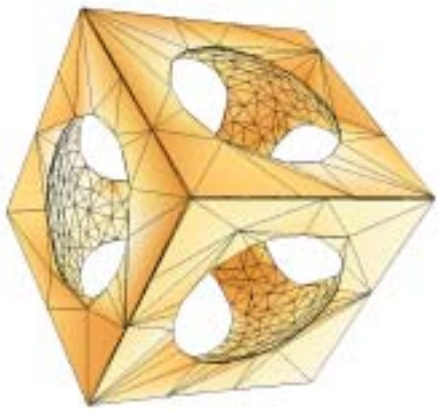


(a) Implicit surface original: 30.832 faces



(b) Implicit surface simplified: 3.852 faces

(c) CSG model original: 39.760 faces



(d) CSG model simplified: 1.240 faces

Figure 17: *Simplification examples*

10. Conclusions and Future works

In this paper we presented the Fast Stellar simplification algorithm, which is a fast implementation for the Four–Face cluster algorithm. We adopted the Corner–Table data structure to represent the surface, which shows to be very suitable not only to obtain a simple implementatio of stellar operators but also for the encoding. We described Stellar Theory, which is powerful mathematical tool for topological mesh operations and pointed out the conditions to safely apply the Edge–Weld and Edge–Flip operators. We also proposed two strategies to encode and decode the hierarchy of surfaces generated by the fast stellar mesh simplification.

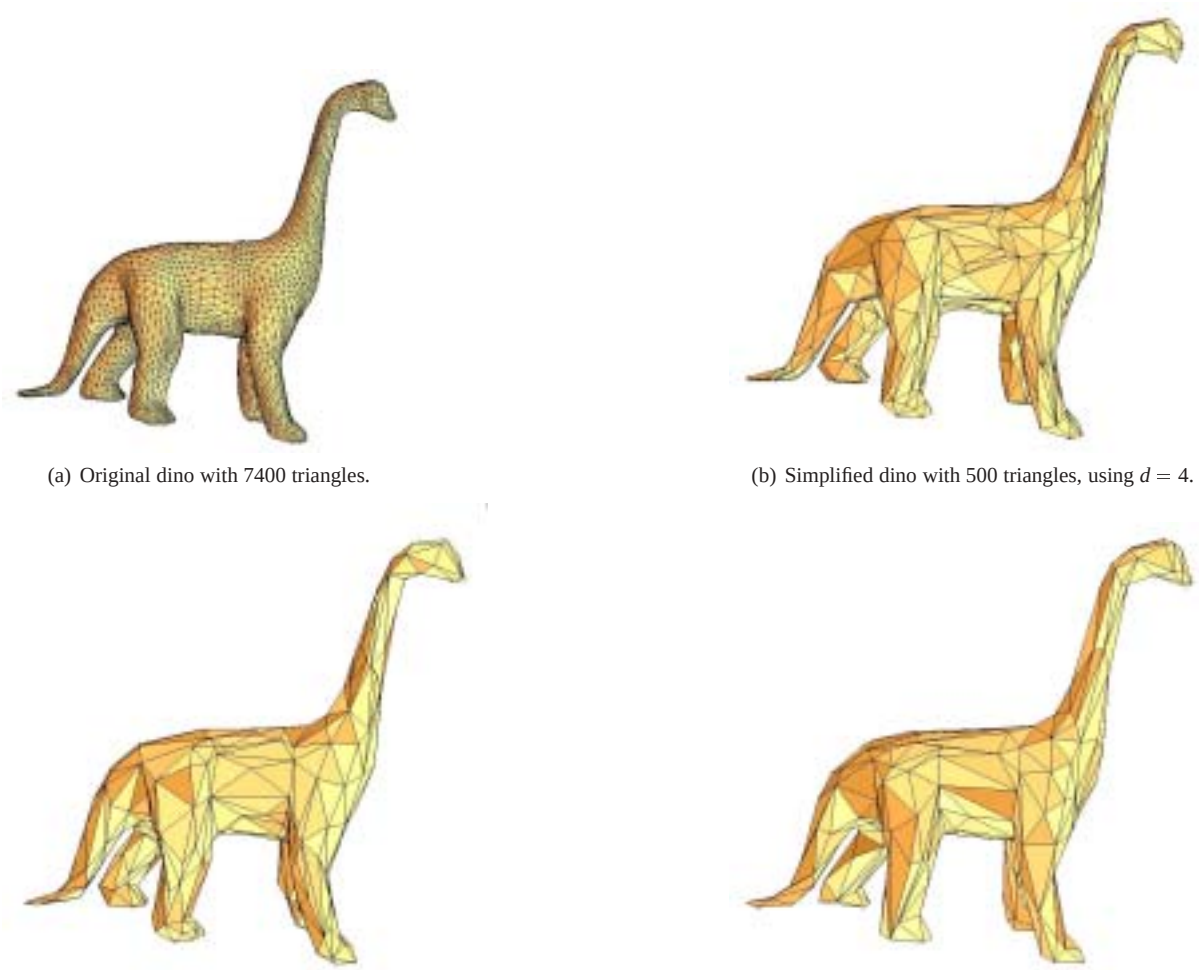
We plan to continue this work in three directions. First, we will try to create an efficient compression scheme for the sequential encoding. Second, we intend to use this simplification algorithm to obtain a parameterization scheme for surfaces, which is a very active area of research in Computer Graphics. Finally, we will investigate an out of core implementation of this algorithm taking advantage of the simplicity of the Corner–Table data structure.

References

[Ale30] ALEXANDER J.: The combinatorial theory of complexes. *Ann. Math. 31* (1930), 294–322. 2, 3, 4

[CMS98] CIGNONI P., MONTANI C., SCOPIGNO R.: A comparison of mesh simplification algorithms. *Computer & Graphics 22*, 1 (1998), 37–54. 1

[Ede02] EDELSBRUNNER H.: *Geometry and Topology for Mesh Generation*. Cambridge Mono-



(a) Original dino with 7400 triangles.

(b) Simplified dino with 500 triangles, using $d = 4$.

(c) Simplified dino with 500 triangles, using $d = 8$.

(d) Simplified dino with 500 triangles, using $d = 12$.

Figure 19: *Dino example.*

graphs on Applied and Computational Mathematics, 2002. 3

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. *Computer Graphics 31*, Annual Conference Series (1997), 209–216. 1, 7

[Gue99] GUEZIEC A.: Locally toleranced surface simplification. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (1999), 168–189. 7

[HDD*93] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Mesh optimization. *Computer Graphics 27*, Annual Conference Series (1993), 19–26. 5, 6

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM Press, pp. 99–108. 1, 9

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM Press, pp. 163–169. 1

[Lic99] LICKORISH W. B. R.: Simplicial moves on complexes and manifolds. In *Proceedings of the Kirbyfest* (1999), vol. 2, pp. 299–320. 3

[LLVT03] LEWINER T., LOPES H., VIEIRA A. W., TAVARES G.: Efficient implementation of marching cubest' cases with topological guaran-

	FFC		FS	
Operation	**t**(s)	**%**	**t**(s)	**%**
Assign Quadrics	0.110	1.26	0.110	1.95
Compute $E(v)$	7.880	90.51	4.930	87.54
Update Queue	0.360	4.14	–	–
Choose d random v	–	–	0.340	6.04
Edge Swaps	0.165	1.90	0.088	1.56
Vertex Removal	0.170	1.95	0.141	2.50
Recompute Q_u/Q_w	0.021	0.24	0.023	0.41
Total	8.706	100	5.632	100

Table 1: *Algorithm's steps comparison.*

	# Triangles		Running time		Ratio
Model	**Input**	**Output**	**FFC**	**FS**	**FS/FFC**
Cow	5804	202	4.680	2.780	0,59
Terrain	5708	190	4.520	2.690	0,59
Torus	14144	344	9.100	5.110	0,56

Table 2: *Total time comparison.*

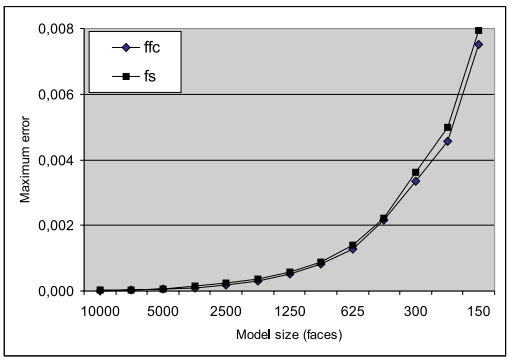


Figure 18: *Geometric error comparison.*

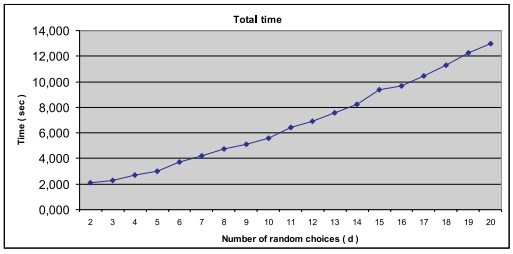


Figure 20: *Total algorithm time as a function of d.*

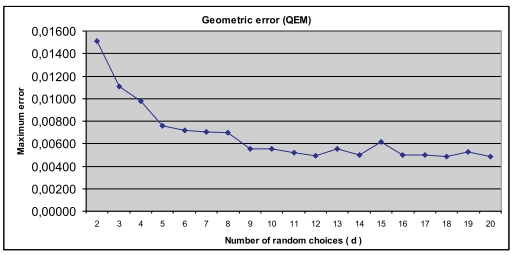


Figure 21: *Geometric error as a function of d.*

tees. *Journal of Graphics Tools 8*, 2 (2003), 1–15. 9

[LRS*02] LOPES H., ROSSIGNAC J., SAFANOVA A., SZYMCZAK A., TAVARES G.: Edgebreaker: a simple compression for surfaces with handles. In *Proceedings of the seventh ACM symposium on Solid modeling and applications* (2002), ACM Press, pp. 289–296. 9

[New26] NEWMAN M. H. A.: On the foundations of combinatorial analysis situs. *Proc. Royal Acad. 29* (1926), 610–641. 2, 3

[Pac91] PACHNER U.: Pl homeomorphic manifolds are equivalent by elementary shellings. *Europ. J. Combinatorics 12* (1991), 129–145. 3

[PS97] PUPPO E., SCOPIGNO R.: *Simplification, LOD and multireolution – principles and applications.* Eurographics 97 Tutorial Press, 1997. 1

[RSS01] ROSSIGNAC J., SAFONOVA A., SZYMCZAK A.: 3d compression made simple: Edgebreaker on a corner-table. In *Proceedings of the 2001 Shape Modeling International Conference* (2001). 2, 4

[Vel01] VELHO L.: Mesh simplification using four-face clusters. In *Proceedings of SMI 2001* (May 2001), Instituto per la Matematica Applicata - CNR, IEEE Computer Society. International Conference on Shape Modeling and Applications. 2, 7, 8

[WK02] WU J., KOBBELT L.: Fast mesh decimation by multiple-choice techniques, 2002. 1, 8, 9

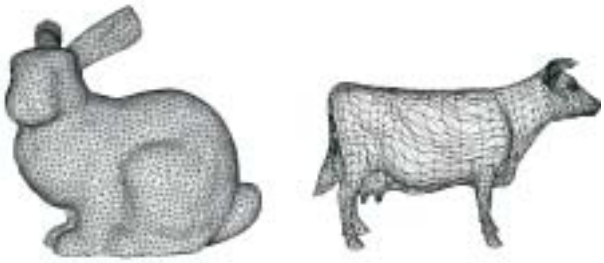
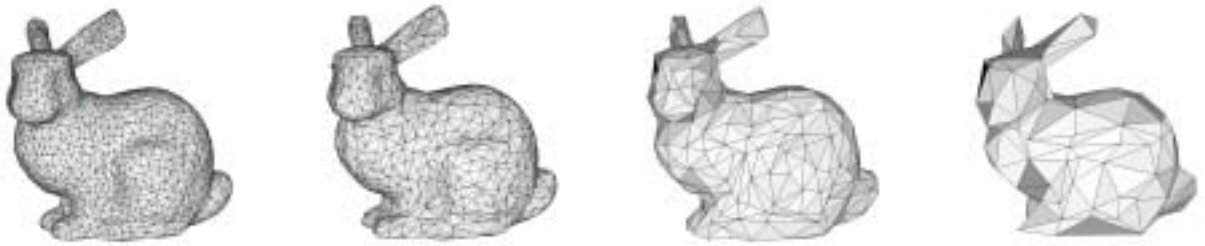


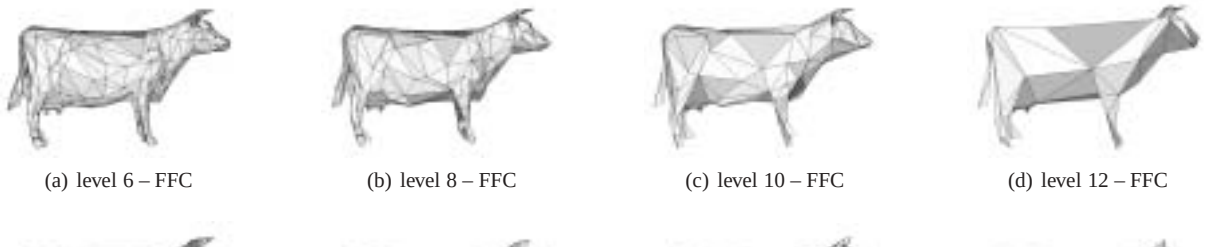
Figure 22: *Models used in the examples: Bunny model (9672 faces) and Cow model (5804 faces).*



(a) level 2 – FFC (b) level 4 – FFC (c) level 8 – FFC (d) level 11 – FFC

(e) level 2 – FS (f) level 4 – FS (g) level 8 – FS (h) level 11 – FS

Figure 23: *Bunny model simplified with 5098, 2628, 696 and 260 faces.*



(a) level 6 – FFC (b) level 8 – FFC (c) level 10 – FFC (d) level 12 – FFC

(e) level 6 – FS (f) level 8 – FS (g) level 10 – FS (h) level 12 – FS

Figure 24: *Cow model simplified with 1182, 640, 382 and 202 faces.*