Fast Stellar Mesh Simplification

ANTÔNIO W. VIEIRA[1,2], LUIZ VELHO[3], HÉLIO LOPES[1], GEOVAN TAVARES[1] AND THOMAS LEWINER[1,4]

[1] PUC–Rio — Laboratório Matmídia — Rio de Janeiro
[2] UNIMONTES — CCET — Montes Claros
[3] IMPA — Laboratório Visgraf — Rio de Janeiro
[4] INRIA — Géométrica Project — Sophia Antipolis
awilson@mat.puc--rio.br, lvelho@impa.br, lopes@mat.puc--rio.br
tavares@mat.puc--rio.br, tomlew@mat.puc--rio.br

Abstract. This paper introduces *Stellar Simplification*, a fast implementation of the Four–Face Cluster algorithm. In our version of this mesh simplification scheme, we adopt a probabilistic heuristic that substitutes the priority queue of the original algorithm. This made our version, in average, 40% faster. In our implementation, we adopt a very concise data structure which uses only two arrays of integers to represent the surface topology. We also introduce a new scheme to encode and decode the hierarchy of meshes generated by the simplification algorithm. This scheme can be used for progressive transmission and compression of meshes.

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 the Marching Cubes [8]. On the other hand, meshes with a high level of detail even without redundancies could be extremely expensive not only for rendering, but also for storing or exchanging them through the web [5]. In both cases, efficient simplification process can get simpler models with lower computational costs [1]. Moreover, the hierarchical multiresolution structure produced by our simplification can be used in a wide range of applications [9].

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 [3]. 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 solve this problem, Wu and Kobbelt [12] 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 d randomly selected edges.

Velho in [10] proposed a new method, called Four–Face cluster mesh simplification, which produces a sequence of *Edge–Weld* operations intercalated with *Edge–Flip* operations. Edge–Weld and Edge–Flip are stellar operators [11]. 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. In this algorithm the energy functions are computed on vertices, and not on 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. For a complete presentation of stellar operators see [11].

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 [6], which allows a concise and simple implementation of our new algorithm. Finally, we propose two practical ways to encode and decode, in a compressed way, the hierarchy of the multiresolution mesh.

Paper outline. Section 2 describes the Corner–Table data structure. Section 3 presents the topological operators that will be used in our algorithm. Section 4 presents the original Four–Face cluster algorithm. Section 5 introduces the Fast Stellar Simplification Algorithm based on the multiple choice technique. Section 6 presents the scheme to encode and decode the hierarchy of the surfaces generated by the simplification algorithm. Finally, Section 7 shows some results and comparison with the former Four–Face cluster algorithm.

2 Corner–Table Data Structure

Combinatorial manifolds. We will consider an orientable triangulated combinatorial surface S. This is the general case of manifold triangle meshes embedded in $\mathbb{R}^3$, but we are only concerned with their connectivity, e.g., the triangle/vertex incidence and the triangle/triangle adjacency information. For more details in the following definitions, see [2].

In a triangle mesh M, the *star* of a vertex v is a subset of M composed by the union of simplices that are incident to v, and is denoted by $star(v)$. The *link* of a vertex v, denoted by $link(v)$, is the frontier of $star(v)$.

Definition 1 (combinatorial surface) *A triangular mesh M is a combinatorial surface if:*

– *Every edge in M is bounding either one or two triangles.*

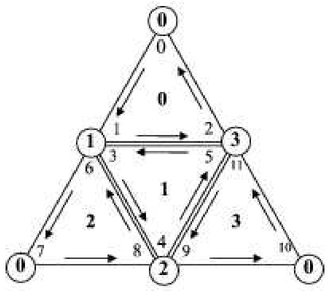
– *The $link$ of a vertex in M is homeomorphic either to an interval or to a circle.*

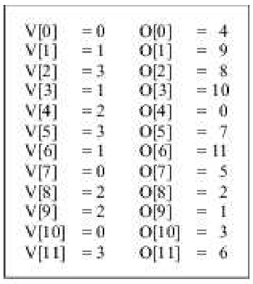
Corner–Table. The Corner–Table is a very concise data structure for triangular meshes. It uses the concept of *corner* to represent the association of a triangle with one of its bounding vertices, or equivalently the association a triangle with its bounding edge opposite to that corner: it may be viewed as a compact version of the half–edge representation of 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 $c.t = c \div 3$. 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: $next(c) = (c + 1) \bmod 3$, and $prev(c) = (c + 2) \bmod 3$.

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 geometrical edge. This information is stored in two integer arrays, called the V and O tables.





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 1: Tetrahedron example.

Example. To illustrate the data structure tables consider the Figure 1. The mesh on the left is a triangulation of a tetrahedron and the table on the right corresponds to its Corner–Table representation.

3 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 $e = (u, v)$, 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, without changing 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.

Let s and t be the vertices opposite to $e = (u, v)$, which is the edge to be collapsed (see Figure 2). Notice that, in this case, the valence of vertices s and t will decrease during the Edge–Collapse operation. Therefore, if s and t are not on the boundary of the mesh, their valence

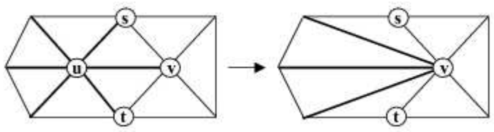


Figure 2: Edge–Collapse.

should be greater than 3 to avoid the creation of a non–manifold object. In practice, choosing edges according to the following *link condition* will guarantee the topological consistency of this operation [2].

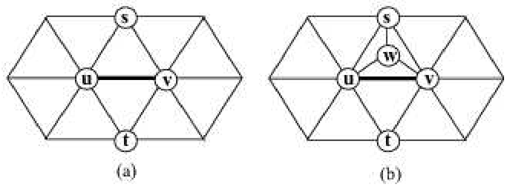


Figure 3: Link Condition.

Lemma 2 *(Link Condition) Let S be a combinatorial 2–manifold. The contraction of an edge $e = (u, v) \in S$ preserves the topology of S if and only if $link(u) \cap link(v) = link(e)$.*

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

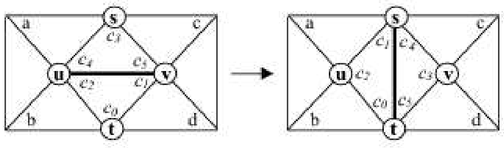


Figure 4: Edge–Flip.

Let the edge $e = (u, v)$ and s and t the two vertices opposed to e (see Figure 4). 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). Notice that the valence of vertices u and v will decrease. If those vertices are internal to the mesh, their initial valence should be greater than 3, otherwise it will generate a valence 2 vertex, breaking the manifold conditions. In the Corner–Table data structure, each edge can be univocally represented by one of its opposite corners. This is used by the Algorithm 1 to perform the *Edge–Flip* operation on the edge opposite to the corner c_0.

Algorithm 1: Edge–Flip(y)

```
// Label incident corners
$c_2 = prev(c_0)$; $c_1 = next(c_0)$;
$d = O[c_1]$; $b = O[c_2]$;
$c_3 = O[c_0]$; $c_4 = next(c_3)$; $c_5 = prev(c_3)$;
$c = O[c_4]$; $a = O[c_5]$;
// 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
$O[c_2] = c_3$; $O[c_0] = a$;
$O[c_3] = c_2$; $O[c_4] = d$; $O[c_5] = c$;
$O[a] = c_0$; $O[c] = c_5$; $O[d] = c_4$;
```

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

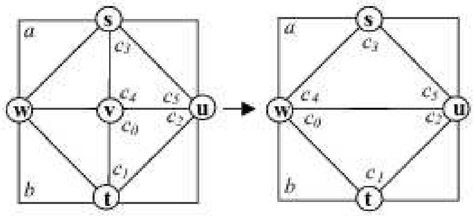


Figure 5: Edge–Weld.

Consider a valence 4–vertex v, adjacent to the vertices s, t, w and u (see Figure 5). 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.

Theorem 3 *Consider a combinatorial 2–manifold M, and a vertex v of M with valence 4. With the notations of Figure 5, 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 *link 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. $\square$

Given a mesh represented by a Corner–Table, the Algorithm 2 performs the removal of the vertex incident to the corner c_0:

Algorithm 2: 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
$V[c_0] = V[O[c_2]]$; $V[c_4] = V[c_0]$;
// Mark removed elements
$O[O[c_2]] = O[O[c_5]] = REMOVED$
$O[next(O[c_2])] = O[prev(O[c_2])] = REMOVED$;
$O[next(O[c_5])] = O[prev(O[c_5])] = REMOVED$;
// Reset opposite corners
$O[c_5] = a$; $O[a] = c_5$;
$O[b] = c_2$; $O[c_2] = b$;
```

Luiz Velho proved in [10] that the *Edge–Collapse* operation can be decomposed into a sequence of *Edge–Flips* operations, followed by one *Edge–Weld* operation. The Figure 6 illustrates this process.

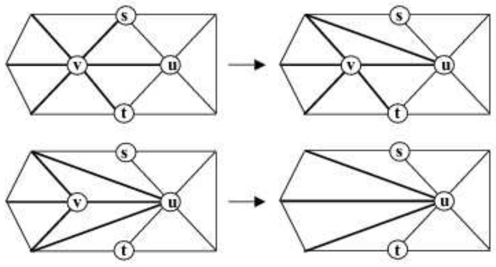


Figure 6: Decomposition of an Edge–Collapse into stellar operators.

4 The Four–Face Cluster Method

The Four–Face Cluster algorithm (FFC) is based on the Edge–Weld and Edge–Flip operators. Since the Edge–Collapse operation is equivalent to a sequence of those stellar operators, the Four–Face Cluster method can implement an Edge–Collapse based method. However, the 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 consequence, the Four–Face Cluster (FFC) algorithm is more flexible than Edge–Collapse based algorithms.

Algorithm's outline. The FFC algorithm naturally 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 levels of resolutions.

In the initialization step for each level of detail j, all the vertices of M^{j-1} are marked as valid for removal. For each level of detail j, we select an unmarked 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 6 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}.

Quadric Error Metric. Each local simplification introduces a geometric distortion between the surfaces M^j and M^{j+1}. The original Four–Face Cluster algorithm used a priority queue to store the vertices to be removed by increasing geometric distortion. This distortion is computed using the QEM [3], as follow.

Let v be a vertex of M^j, and $f_i \in star(v)$ 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 [3] 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 $d\left(w\right)$ of the point w to the plane p_i:

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

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

Simplification criterion. The FFC algorithm assigns an error metric Q_v to each vertex v of the mesh, which is computed as the weighted sum of the distance to the faces f_i:

$$Q_v = \sum a_i Q_i$$

We compute the error introduced by removing a vertex v from the mesh, considering each Edge–Flip and Edge–Weld costs, by [10]:

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

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 [4] and trying to preserve the dihedral angle (angle between the normals of the incident faces of the edge candidate for the swap).

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 Algorihtm 3 gives the pseudo–code for the Four–Face Cluster Simplification Algorithm introduced in [10]. It generates n levels of resolution of a given mesh M.

Algorithm 3: SimplifyFFC(M, n)

```
assign quadrics;
for all($v \in M$) 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 $w \in link(v)$
      remove vertex $(v)$
      re–compute the errors $Q_u$ and $Q_w$
      update queue for $w \in link(u) \cup link(w)$
```

5 The Fast Stellar Simplification Algorithm

An alternative to simplification methods based on priority queue was presented by Wu and Kobbelt in [12]. 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 3D model 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:

$$\left(\frac{5}{100}\right)^8 \approx 10^{-1}$$

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 4 shows the pseudo–code to generate n levels of resolution for a mesh M using this process:

Algorithm 4: 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 $w \in link(v)$
    remove vertex $(v)$
    re–compute quadrics $Q_u$ and $Q_w$
```

6 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 [7]. We modify this implementation by adding a fourth attribute to vertex geometry, its index on the G table of the original surface. Mantaining 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 [5], 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 7 illustrates a vertex simplification with the encoding of the corresponding operations.

Let call v the vertex to be removed, an $e = (u, v) \in star(v)$ 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.

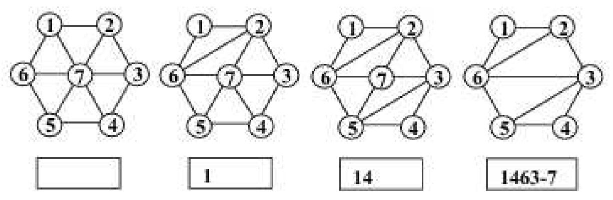


Figure 7: Encoding of a vertex simplification.

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 [5], the starting surface for refinement is restricted to be the base mesh.

7 Results

The results presented in [12] 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.

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. Notice that the total time for the Stanford's Bunny model in Table 1 is greater then in Table 2 because of we did some more measures of time spent by each routine.

In Figure 8 we present two examples of models to visually compare the two algorithms. Notice that, as in [12], the results got in both methods are similar.

For the first example, we chose the Stanford's Bunny model. Figure 9 shows the levels of detail $2, 4, 8$ and 11 obtained using the FFC and the FS.

Figure 10 shows the simplification of *cow* model, our second example. We show in this figure the levels of detail $6, 8, 10$ and 12obtained using the FFC and the FS algorithms.

8 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 for the stellar operators' implementation but

	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: Time comparison of each algorithm's s.ps.

	# Triangles		Running time		Ratio
Model	**Input**	**Output**	**FFC**	**FS**	**ffc/fs**
Bunny	9672	260	7.320	4.130	1.77
Cow	5804	202	4.680	2.780	1.68
Terrain	5708	190	4.520	2.690	1.68
Torus	14144	344	9.100	5.110	1.78

Table 2: Total time comparison.

also for the encoding. We 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 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 a hardware implementation of this algorithm taking advantage of the simplicity of the Corner–Table data structure.

References

[1] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computer & Graphics*, 22(1):37–54, 1998.

[2] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge Monographs on Applied and Computational Mathematics, 2002.

[3] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics*, 31(Annual Conference Series):209–216, 1997.

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

[5] H. Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM Press, 1996.

[6] J. Rossignac, A. Safonova, and A. Szymczak. 3D Compression Made Simple: Edgebreaker on a Corner-Table. In *Proceedings of the 2001 Shape Modeling International Conference*, 2001.

[7] H. Lopes, J. Rossignac, A. Safanova, A. Szymczak, and G. Tavares. Edgebreaker: a simple compression for surfaces with handles. In *Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 289–296. ACM Press, 2002.

[8] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.

[9] E. Puppo and R. Scopigno. *Simplification, LOD and multireolution – principles and applications*. Eurographics 97 Tutorial Press, 1997.

[10] L. Velho. Mesh simplification using four-face clusters. In *Proceedings of SMI 2001*. Instituto per la Matematica Applicata - CNR, IEEE Computer Society, May 2001. International Conference on Shape Modeling and Applications.

[11] L. Velho, H. Lopes, and G. Tavares. Mesh ops. *Manuscript submitted to publication*, 2003.

[12] J. Wu and L. Kobbelt. Fast mesh decimation by multiple-choice techniques. In *Vision, Modeling, and Visualization 2002*. IOS Press, 2002.

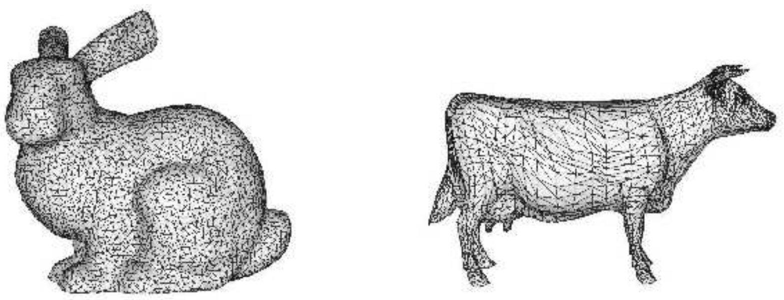
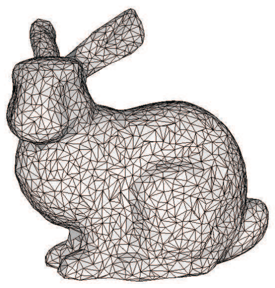
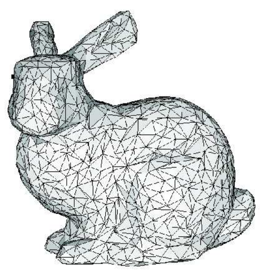


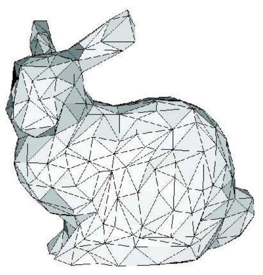
Figure 8: Original models used in the examples: Bunny model with 9672 faces(left) and Cow model with 5804 faces (right).



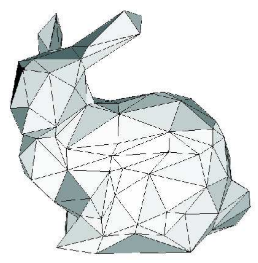
(a) level 2 – FFC



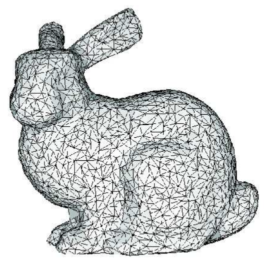
(b) level 4 – FFC



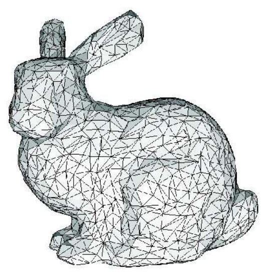
(c) level 8 – FFC



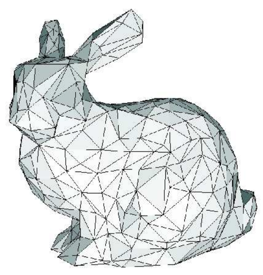
(d) level 11 – FFC



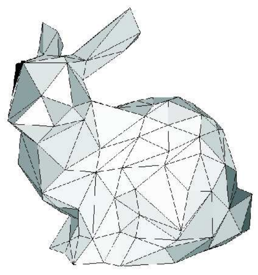
(e) level 2 – FS



(f) level 4 – FS

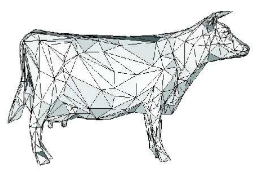


(g) level 8 – FS

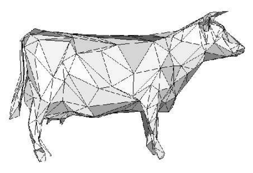


(h) level 11 – FS

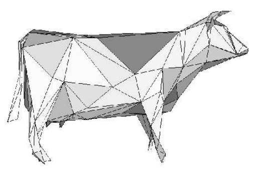
Figure 9: Bunny model simplified with 5098, 2628, 696 and 260 faces.



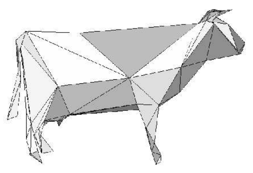
(a) level 6 – FFC



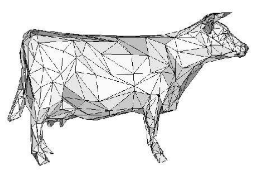
(b) level 8 – FFC



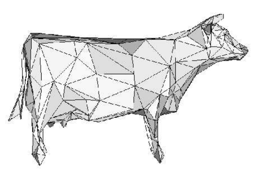
(c) level 10 – FFC



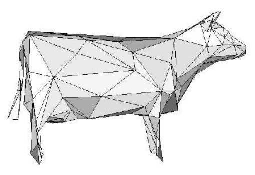
(d) level 12 – FFC



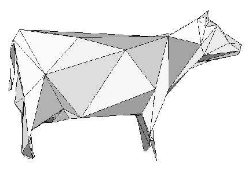
(e) level 6 – FS



(f) level 8 – FS



(g) level 10 – FS



(h) level 12 – FS

Figure 10: Cow model simplified with 1182, 640, 382, 302 and 202 faces.