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.

Keywords: Mesh simplification. Geometry processing. Randomized algorithms.

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 \cite{Lorensen1987}. 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 \cite{Wilson2001}. In both cases, efficient simplification process can get simpler models with lower computational costs \cite{Wilson2001}. Moreover, the hierarchical multi–resolution structure produced by our simplification can be used in a wide range of applications \cite{Wilson2001}.
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 \[ E(Q, W) = \sum_{v \in V} \frac{1}{2} \left( Q^2 - 2GWQ + GW^2 \right) \] for a mesh \( Q \) and a given weight matrix \( W \).

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 \[ 10 \].

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 \[ 8 \], 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 multi–resolution mesh.

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

2 Corner–Table Data Structure

Basic concepts. A simplex \( \sigma^p \) of dimension \( p \) (p-simplex, for short) is the convex hull of \( p + 1 \) linearly independent points in \( \mathbb{R}^m \), called its vertices. A p-simplex has \( p \) dimensions and is composed of \( p + 1 \) simplices of dimension \( (p – 1) \), called faces. The set of 2, 1 and 0 dimensional simplices will be called, respectively, triangles, edges and vertices. If \( \sigma \) is a subsimplex of a simplex \( \tau \) then they are said to be incident to each other.

A simplicial complex \( \mathcal{C} \) is a finite set of simplices together with all its subsimplices such that if \( \sigma^k \) and \( \tau^p \), \( k \leq p \) belong to \( \mathcal{C} \), then either \( \sigma^k \) and \( \tau^p \) meet at a subsimplex \( \lambda^m \), \( m \leq k \), or are completely disjoint. A triangle mesh is a simplicial complex whose simplices are at most dimension 2.

Combinatorial manifolds. We will consider an orientable triangulated combinatorial surface \( \mathcal{S} \). 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 \( \text{star}(v) \). The link of a vertex \( v \), denoted by \( \text{link}(v) \), is the frontier of \( \text{star}(v) \), or the union of 0 and 1 simplices of \( \text{star}(v) \) that are not incident to \( v \). The Link of an edge \( (u,v) \) are the opposite vertices by the edge \( (u,v) \).

Definition 1 (combinatorial surface) A triangular mesh \( M \) is a combinatorial surface if: Every edge in \( M \) is bounding either one or two triangles and 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 \( c.t = \text{floor}(c / 3) \).

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:

\[
\text{next}(c) = 3c.t + (c + 1) \mod 3, \quad \text{prev}(c) = 3c.t + (c + 2) \mod 3.
\]

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. Figure 2 shows an example of a Corner–Table representation for a tetrahedron.

![Tetrahedron example](image1)

**Figure 2: Tetrahedron example**

### 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 \( \tau \). 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 \( \tau \) 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 3). Choosing edges according to the following link condition will guarantee the topological consistency of this operation [2] (see Figure 4).

![Corner–Table](image2)

**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 \( \text{link}(u) \cap \text{link}(v) = \text{link}(e) \).

**Edge–Flip:** This operation consists in transforming a two–face cluster into another two–face cluster by swapping its common edge. Let the edge \( e = (u, v) \) and \( s \) and \( t \) the two vertices opposed to \( e \) (see Figure 5). The Edge–Flip operation will replace \( e \) by \( (s, t) \), and replace the 2 triangles incident to \( e \) by \( (u, s, t) \) and \( (v, t, s) \). 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 \( c_0 \).

![Corner–Table](image3)

**Algorithm 1: Edge–Flip\((c_0)\)**

// Label incident corners
\[
\begin{align*}
e_2 &= \text{prev}(c_0); && c_1 &= \text{next}(c_0); \\
c_3 &= O[c_0]; && c_4 &= \text{next}(c_3); && c_5 &= \text{prev}(c_3); \\
a &= O[c_3]; && b &= O[c_2]; && c &= O[c_4]; && d &= O[c_1];
\end{align*}
\]

// Label incident vertices
\[
\begin{align*}
l &= V[c_0]; && v &= V[c_1]; && s &= V[c_3];
\end{align*}
\]

// Perform swap
\[
\begin{align*}
V[c_1] &= s; && V[c_3] &= v; && V[c_4] &= s; && V[c_5] &= t;
\end{align*}
\]

// Reset opposite corners
\[
\begin{align*}
O[e_2] &= c_3; && O[c_0] &= a; && O[c_1] &= c_2; && O[c_4] &= d;
\end{align*}
\]

Preprint MAT. 14/03, communicated on March 3rd, 2003 to the Department of Mathematics, Pontifícia Universidade Católica — Rio de Janeiro, Brazil.
**Edge–Weld.** This operation consists in transforming a four–face cluster into a two–face cluster by removing its central vertex. Consider a valence $4$–vertex $v$, adjacent to the vertices $s$, $t$, $w$ and $u$ (see Figure 6). 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 6 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: $\text{link}(v) \cap \text{link}(u) = \{s,t\} = \text{link}(e)$. As $\{u,w\} = \text{link}(v) \setminus \text{link}(e)$, this condition is valid if and only if $w$ does not belong to $\text{link}(u)$. This means that there is no edge connecting $w$ to $u$. □

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($c_0$)**

```plaintext
// Assign incidences
$c_1 = \text{next}(c_0)$; $c_2 = \text{prev}(c_0)$;
$c_4 = \text{next}(\text{O}[c_1]);$ $c_5 = \text{prev}(\text{O}[c_1]);$
$a = \text{O}[\text{next}(\text{O}[c_5])];$ $b = \text{O}[\text{prev}(\text{O}[c_2])];$

// Perform vertex removal
$V[c_0] = V[O[c_2]]; V[c_4] = V[c_0];$
// Reset opposite corners
$O[c_5] = a;$ $O[a] = c_5;$ $O[b] = c_2;$ $O[c_2] = b;$
```

Luiz Velho proved in [9] that the Edge–Collapse operation can be decomposed into a sequence of Edge–Flips operations, followed by one Edge–Weld operation. The Figure 7 illustrates this process.

**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 constructs a hierarchy of meshes $(M^0, M^1, \ldots, M^n)$ 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. 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 7 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.

**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 [3], as follow.
Let $v$ be a vertex of $M^j$, and $f_i \in \text{star}(v)$ 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 [13] by:

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

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

$$d_i(w) = w^T (p_ip_i^T) w = w^T Q_i w$$

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

**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 [9]:

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

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:

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

The cost $S(v)$ is the sum of the cost of Edge–Flips in $\text{Star}(v)$ used to bring $v$ to valence 4. To compute this cost, we compute the sequence of independent Edge–Flips, minimizing the aspect ratio [12] 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 3 gives the pseudo–code for the Four–Face Cluster Simplification Algorithm introduced in [9]. It generates $n$ levels of resolution of a given mesh $M$.

---

**Algorithm 3: SimplifyFFC($M, n$)**

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

---

5 The Fast Stellar Simplification Algorithm

An alternative to simplification methods based on priority queue was presented by Wu and Kobbelt in [11]. 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: $(\frac{5}{100})^8 \approx 4 \times 10^{-11}$.

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 recompute 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:
In order to represent the hierarchy of meshes, we use a Multi–resolution representation. This strategy produces a progressive mesh, whose resolution changes globally at each level.

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 [4], 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 8 illustrates a vertex simplification with the encoding of the corresponding operations. Let call $v$ the vertex to be removed, and $e = (u, v) \in \text{star}(v)$ 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 [2], the starting surface for refinement is restricted to be the base mesh.

7 Results

The results presented in [11] 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 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. No-
**Fast stellar mesh simplification**

<table>
<thead>
<tr>
<th>Operation</th>
<th>FFC</th>
<th>%</th>
<th>FS</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign Quadrics</td>
<td>0.110</td>
<td>1.26</td>
<td>0.110</td>
<td>1.95</td>
</tr>
<tr>
<td>Compute $E(v)$</td>
<td>7.880</td>
<td>90.51</td>
<td>4.930</td>
<td>87.54</td>
</tr>
<tr>
<td>Update Queue</td>
<td>0.360</td>
<td>4.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Choose a random vertex</td>
<td></td>
<td></td>
<td>0.340</td>
<td>6.04</td>
</tr>
<tr>
<td>Edge Swaps</td>
<td>0.165</td>
<td>1.90</td>
<td>0.088</td>
<td>1.56</td>
</tr>
<tr>
<td>Vertex Removal</td>
<td>0.170</td>
<td>1.95</td>
<td>0.141</td>
<td>2.50</td>
</tr>
<tr>
<td>Recompute $Q_u/Q_w$</td>
<td>0.021</td>
<td>0.24</td>
<td>0.023</td>
<td>0.41</td>
</tr>
<tr>
<td>Total</td>
<td>8.706</td>
<td>100</td>
<td>5.632</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 1: algorithm’s steps comparison.

<table>
<thead>
<tr>
<th>Model</th>
<th># Triangles</th>
<th>Running time</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bunny</td>
<td>9672</td>
<td>7.320</td>
<td>1.77</td>
</tr>
<tr>
<td>Cow</td>
<td>5804</td>
<td>4.680</td>
<td>1.68</td>
</tr>
<tr>
<td>Terrain</td>
<td>5708</td>
<td>4.520</td>
<td>1.68</td>
</tr>
<tr>
<td>Torus</td>
<td>14144</td>
<td>9.100</td>
<td>1.78</td>
</tr>
</tbody>
</table>

Table 2: Total time comparison.

Notice that the total time for the Stanford’s Bunny model in Table 1 is greater than in Table 2 because of some more measures of time spent by each routine. Figure 3 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 4 we present two examples of models to visually compare the two algorithms. Notice that, in Table 2, the results got in both methods are similar. For the first example, we chose the Stanford’s Bunny model. Figure 5 shows the levels of detail 2, 4, 8 and 11 obtained using the FFC and the FS. Figure 6 shows the simplification of cow model, our second example. We show in this figure the levels of detail 6, 8, 10 and 12 obtained using both 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 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**


Preprint MAT. 14/03, communicated on March 3rd, 2003 to the Department of Mathematics, Pontifícia Universidade Católica — Rio de Janeiro, Brazil.
Figure 10: Models used in the examples: Bunny model (9672 faces) and Cow model (5804 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 11: Cow model simplified with 1182, 640, 382 and 202 faces.


