Octree-based view-dependent triangle meshes

Marta Fairén
Department of Software
Universitat Politècnica de Catalunya
mfairen@lsi.upc.edu

Ramón Trueba
Department of Software
Universitat Politècnica de Catalunya
rtrueba@lsi.upc.edu

ABSTRACT

In this paper we present a new technique for view-dependent LOD rendering, where the scene is represented through an octree model from which we can obtain a triangle mesh corresponding to a view-dependent LOD. We present the construction of this octree model and the visualization algorithm that generates on-the-fly a closed and valid triangle mesh for each frame of the visualization. This visualization algorithm is a depth-first traversal algorithm which also allows to re-use triangles from one frame to another.

Keywords: View-dependent LOD rendering, octree-based model, valid triangle mesh generation.

1 INTRODUCTION

During the last few decades substantial research efforts have been devoted to devise new techniques to provide interactive navigation through complex 3D models containing thousands of objects and millions of primitives. Most recent approaches fall into three major techniques: level-of-detail (LOD) rendering, image-based rendering, and occlusion culling.

In the case of level-of-detail rendering and more concretely in view-dependent LOD rendering the main problem these techniques have is in those regions where there is a change of level among neighbours, where cracks in the generated triangulation should be avoided.

In our proposal the scene is represented through an octree model which is able to differentiate among objects (allowing further operations like selection, for example, during the navigation) and from which we can obtain a triangle mesh corresponding to a view-dependent LOD. We present the construction of this octree and the algorithm that generates on-the-fly, by doing a depth-first traversal of the octree, a closed and valid triangle mesh for each frame of the visualization.

Some differences between our proposal and other view-dependent LOD techniques are: our data structure does not keep topology information; we do not impose any restriction on the difference of levels among neighbour nodes; and we do keep information about objects.

The main contributions of this paper include:

- A new octree-based data structure which encodes the geometry of the scene and which enables the on-line extraction of arbitrary view-dependent LODs.
- An off-line algorithm for building such an octree from different input data.
- An efficient algorithm for rendering the scene encoded by the octree. Triangles extracted from previous frames are efficiently cached and reused on a hierarchical basis.

The rest of the paper is organized as follows. Section 2 briefly reviews the previous work. In section 3 we introduce some definitions and an outline of the algorithm. Sections 4 and 5 explain respectively the definition and generation of the octree model. Section 6 presents the view-dependent octree traversal algorithm. Finally in section 7 we show some results and discussion and we conclude in section 8.

2 PREVIOUS WORK

There are several papers that use hierarchical models to represent very complex scenes and which at the end visualize triangles. Among many of them we can cite [7], which computes a topology preserving isosurface from a volumetric grid using Marching Cubes for geometry processing applications; and [2], which describes an efficient technique for out-of-core construction and accurate view-dependent visualization of very large surface models. It uses a regular conformal hierarchy of tetrahedra to spatially partition the model.

Our approach uses a hierarchical model as well. Similarly to [3], which uses a new spanned sub-meshes simplification technique to build view-dependence trees I/O-efficiently, preserving the correct edge collapsing order and thus assuring the run-time image quality, our octree is view-dependent and we visualize the triangles

 ISBN 978-80-86943-98-5

that correspond to a front which is being adapted to the point of view at each frame.

Another related paper is [5], which describes a new method for contouring a signed grid whose edges are tagged by Hermite data. They develop an octree-based method for simplifying contours produced by this method. Their method generates quads from Black-White edges. As we will see in the following sections in our case we also consider White-White edges as well as those joining different levels among nodes (T-edges).

Our extraction of triangles from edges is similar to how [6] does to generate triangle strips from the dual mesh.

Schmitt [8] also generates triangles from an octree but in his case the front only moves through complete levels (all nodes in the front should be at the same level).

To the best of our understanding, our approach is the first one that includes a complete view-dependent triangle mesh generation from an octree by doing it on-the-fly and allowing the reuse of geometry already computed.

3 DEFINITIONS AND OUTLINE OF THE VISUALIZATION ALGORITHM

The first step of our proposal consists on a pre-process of input data dedicated to generate the structure that will be used by the view-dependent algorithm. Our input data can be a BRep model or a voxel model. From the input data, the pre-process generates an octree where each grey node contains the coordinates of a point representing the surface inside the node, a normal vector and a colour. Each octree node has binary information of their vertices (*in* or *out*). *Out* vertices will be labeled as white whereas *in* vertices will be labeled as black. The octree specification and generation process is further explained in next section. We will start first by some definitions:

Definition 1 *Relevant edge*

An edge of a node is relevant when it has a white vertex and a black vertex or both vertices are white but the original surface intersects the edge.

Definition 2 *Membrane node*

A membrane node is an octree node containing at least a relevant edge.

Definition 3 *Surface edge*

A relevant edge e is a surface edge of an octree node n ***iff*** *e is one of the 12 edges of n.*

Definition 4 *Edge belongs to node*

A relevant edge e belongs to an octree node n ***iff*** *e is part of any surface edge of n.*

As an example, in figure 1(b) we will say that the red edge is a surface edge of nodes B and D and also belongs to nodes A and C.

Definition 5 *Edge belongs to subtree*

Being S a subtree of the octree and e a relevant edge, e belongs to S ***iff***

$$\forall k : e \text{ belongs to } n_k : n_k \text{ belongs to } S$$

(e belongs to S when the two or four nodes having e as an edge are in the subtree S).

Note: If e belongs to a subtree S, e also belongs to all subtrees containing S.

Definition 6 *Edge internal to node*

A relevant edge e is internal to an octree node n ***iff*** *e belongs to one or more descendants of n but it does not belong to n.*

Definition 7 *T-edge*

A T-edge is a relevant edge which belongs to two membrane nodes and also lays over a face of a third node (see figure 1(a)).

Definition 8 *X-edge*

An X-edge is a relevant edge which belongs to four membrane nodes (see figure 1(b)).

Once we have the octree pre-computed, the algorithm presented in section 6 traverses this structure in a view-dependent manner, by using an *on-the-fly* generation of triangles. The algorithm traverses the octree and for each node being processed it computes the required level in order to decide if the octree level of this node is enough, depending on the point of view, to represent the node in this frame as a leaf node.

Definition 9 *Front*

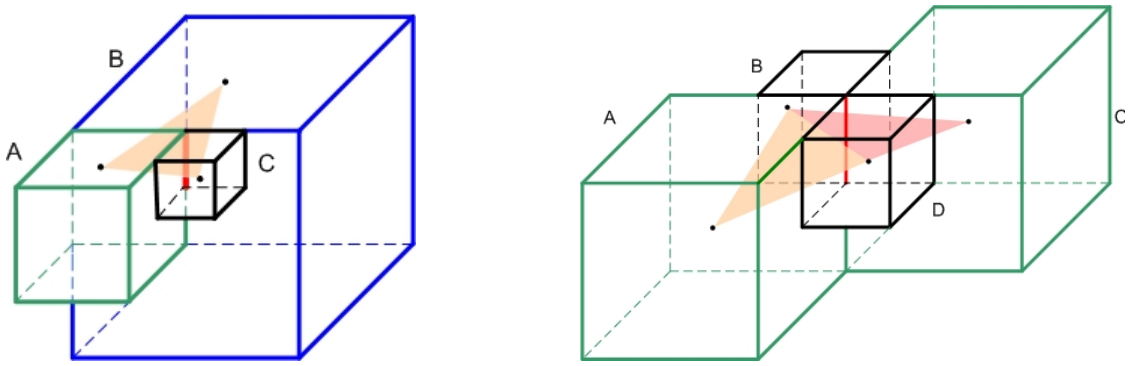
We call front to the set of membrane nodes being at their required level.

The *front* is changing at each frame depending on the point of view. The nodes participating on this front are a subset of the membrane nodes. The view-dependent depth-first traversal of the octree stops when a membrane node is in its required level and this node becomes part of the front for this frame.

A white face of a cube representing a membrane node is a face shared with a white node. The union of all white faces of the nodes of the front will be called the white surface of the front.

Taking all *T-edges* and *X-edges* of those nodes participating on the front and generating the corresponding triangles (one for each T-edge and two for each X-edge), it can be shown that we obtain a valid triangulation which represents the scene at the suitable level of detail, where a valid triangulation is a triangulation that is homeomorphic to the white surface of the front.

 ISBN 978-80-86943-98-5



(a) T-edge (b) X-edge

Figure 1: Example of a T-edge (a) and an X-edge (b). In both cases the edge is the one drawn in red. A T-edge (a) generates a triangle of the final mesh whereas an X-edge (b) generates two triangles. The nearest point to the relevant edge is chosen to divide the quad in the two triangles.

4 OCTREE DEFINITION

The octree model [1] we generate to represent the original model stops its subdivision process when a node does not contain model surface (the node is completely inside or outside the solid); when the node, at this level, already has the necessary information to represent the model surface; or when the subdivision level reaches the maximum level predefined.

With this kind of construction we have a non-restricted octree, where each leaf node can have as neighbours other leaf nodes without any restriction on their size.

For our algorithm, we only need to store information in the grey nodes of the octree, not in the black or white ones. This information includes:

- **A representative point** of the surface inside the node, being represented by its coordinates, a normal vector and a colour.
- **The vertex signs** of the node, indicating, for each one of the 8 vertices of the cube representing the node, whether they are *in* or *out* of the solid.
- **Relevant edges** of the node. We store boolean information indicating which of the 12 edges of the cube are relevant.
- **Object identifier**. We store information about the object whose point is used as a representative point. It facilitates to work with multi-objects.

5 OCTREE GENERATION

Our input data can be a correct BRep model or a voxel model. From this input data, we generate the non-restricted octree containing all the information.

The pre-process consists of two steps. The first step is used to fill the information in the leaf nodes and the second to fill the octree inner nodes. The first step is different depending on the input data.

5.1 Construction from a BRep model

The construction of the octree from a BRep model is done recursively starting from a cube (root node) which includes all the scene, and following a classical subdivision process. This subdivision process stops when the node has the minimum edge length (minimal resolution node) or when the node information fulfills a certain condition (terminal octree node).

The octree nodes are terminal (no more subdivision to do) in two cases:

- when the node doesn't have surface inside it, in this case the node becomes white or black depending on the situation of the node (*in* or *out*) with respect to the solid;
- when the node has solid surface inside it, it becomes terminal if the surface it contains only belongs to an object and fulfills one of the following conditions:
  - it has only one vertex of the geometry;
  - it has only one edge of the geometry;
  - it has only one face of the geometry.

In case the subdivision stops because of a minimal resolution we can have geometry of different objects inside it, so we choose the most relevant object in the node (the one with the largest area inside the node).

The information to be kept in any membrane node is computed as follows:

Representative point coordinates. If the node contains a feature vertex of the solid surface we store

 ISBN 978-80-86943-98-5

the vertex; otherwise, if the node contains a feature edge, we compute that point on the edge nearest to the center of the node; in any other case we compute the nearest point of the surface to the center of the node.

Representative point normal vector. The normal vector stored is the normal of the solid surface at the point we kept as a representative point.

Representative point colour. We take the colour that the representative point has in the original solid.

Signs of vertices. For each node vertex we compute whether it is inside or outside of the solid. The process to compute the signs of vertices is done after the construction of the octree. For each vertex of the cube representing the octree node we compute whether it is white (outside the solid) or black (inside the solid).

Relevant edges. An edge of the node is considered relevant if it has Black-White vertices or it has White-White vertices and the solid surface intersects the edge. This is computed also at the end of the octree construction. We identify those edges having White-White vertices that have to be relevant because the solid surface intersects them. It is also possible to have Black-Black edges intersected by the solid surface, but these edges are not considered relevant because this geometry intersecting the edge is considered internal to the solid and is not visualized.

Object identifier. Identifier of the object whose point is used as a representative point.

5.2 Construction from a voxel model

The construction of the octree from a voxel model starts by identifying each voxel as black, white or membrane node at the minimal resolution of the octree.

The information to be kept in any membrane node at this minimal resolution is filled as follows:

Representative point coordinates. The representative point chosen could be computed through an interpolation from the voxels in the 26-neighbourhood. However, in the present implementation we have chosen to use the middle point of the voxel, because its size is sufficiently small.

Representative point normal vector. We take the gradient of the voxel calculated from the 26-neighbour intensity.

Representative point colour. The colour will be the one kept in the voxel model or computed by using a transfer function from the value of the voxel model [4].

Signs of vertices. We classify a vertex as interior when this vertex does not have any white node around and we classify it as outer when the vertex touches at least one white node.

Relevant edges. An edge of the node is considered relevant if it has Black-White vertices.

Object identifier. Identifier of the object whose point is used as a representative point.

Once we have identified the minimal resolution nodes, we compact those black/white nodes which can be grouped into other black/white nodes to get lower level terminal octree nodes.

5.3 Filling inner nodes

Once all the leaf nodes are processed we have to fill in the octree inner nodes. We apply a bottom-up simplification algorithm and fill each inner membrane node information as follows:

- The sign of each node's vertex corresponds to the sign of the child sharing this vertex with the node.
- The relevant node's edges are those constituted by two White-Black edges or by two White-White edges with at least one of them being relevant.
- The vertex information, normal, colour and object identifier are taken from one of the children elected as representative following the next criterion: A vote is made among the children to discover which object appears in more nodes (the object with a greater surface inside the node) and among the children voting for that object, the one that has the nearest point to the node's center is chosen as the representative.

6 VIEW-DEPENDENT OCTREE TRAVERSAL

The algorithm we use to traverse the octree and generate the view-dependent triangle mesh can be summarized as follows:

```
1   advance = true;
2   node = first_son(root);
3   while node != root do
4       if not in_frustum(node) then
5           process_edges_node();
6       else
7           if reusable(node) then
8               Reuse_triangles();
9           else
10              if level_required > current_level and
11                 not node.is_terminal() then
12                  node = node.first_son(); // go down
13                  advance = false;
14              else
15                  process_edges_node();
16              endif
17          endif
18      endif
19      if advance then
20          while node != root and
21             node.is_last_son() do
22              node = parent(node); // going up
23              process_edges_non_terminal_node();
24          endwhile
25          if node == root then
26              process_edges_root();
27          else
28              node = next_brother(node);
29          endif
30      endif
31  endwhile
```

The geometry drawn by the algorithm at each frame is computed at run time. The algorithm traverses the octree model and depending on the point of view decides which nodes and at which level will generate the triangles to be drawn (see figure 2).

In order to take profit from the coherence existing between one frame and the next one, the algorithm keeps memory of a list of triangles that stores the triangles drawn in one frame and sorted in the order in which those triangles were generated. This order in the list allows to know each sub-list of triangles belonging to each intermediate node of the octree. When the algorithm decides that a node keeps the same level-of-detail from one frame to the next one, it does not re-compute the triangles for the sub-tree of this node but re-uses those triangles in the list already computed in some previous frame (lines 7 and 8 in the algorithm).

We decide a node is reusable (line 7 in the algorithm) if it fulfills the following conditions:

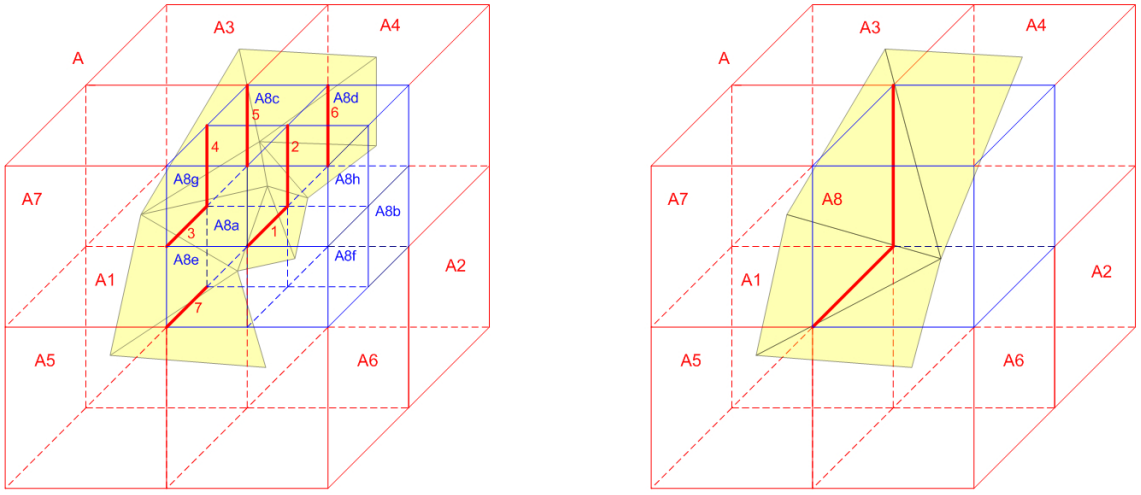
- it has already generated its corresponding triangles in a previous frame (so these triangles are kept in the list of triangles);
- its level-of-detail has not changed from the previous frame;
- the number of frames passed since the last time the triangles of the node were generated does not exceed a certain predefined limit.

The required level for a node being processed on a frame is computed taking into account the number of pixels that the representation of this node is going to cover on its projection from the user point of view. The level required for the front in a certain subtree depends on the apparent size of their nodes in the screen, which should be smaller than a certain given tolerance. A node being near to the user would require a higher level while a node being far from the user would require a lower level (see figure 3).

The depth traversal of the octree has a treatment done to each node while descending and another done while ascending. While descending, a node can be classified and treated as:

- A node that *can be re-used* from the last frame. The list of triangles corresponding to this node are directly re-used from the last frame (line 8 in the algorithm).
- A node considered *at the required level* for the point-of-view. This node is considered a leaf node (in this frame) and the only treatment required for this node is to recognize those edges of the node that are relevant edges and that should be processed in the treatment to do while ascending (line 15 in the algorithm). These relevant edges will be passed to the parent node to be considered by it (see below the explanation of the process done in the intermediate nodes while ascending).
- A node *not being in the frustum*. Although no treatment would be needed for this node, their edges should be considered in order to generate triangles joining this node with others inside the frustum. It is treated as a node being at the required level (line 5 in the algorithm).
- A node whose required level classifies it as *an intermediate node*. These nodes require no treatment while descending, the algorithm just goes down in the tree (lines 12 and 13 in the algorithm).

While ascending in the depth traversal of the octree, the treatment is mainly centered in the intermediate nodes. Each intermediate node has to process those relevant edges that are internal to the subtree represented by the node (it is the root of the subtree), and which have not been processed before, i.e. it processes those relevant edges that are internal to this node and not to any one of its sons (lines 22 and 23 in the algorithm). The relevant edges that are not internal to the subtree represented by the node will be passed to the parent node. In figure 2(a) the relevant edges 1 and 2 will be



(a) (b)

Figure 2: Triangle mesh generated for an example having 7 relevant edges (a) (relevant edges are drawn in thick red line). In case the node A8 is considered at the required level of subdivision (b) only 2 relevant edges are considered and the triangle mesh is simplified.

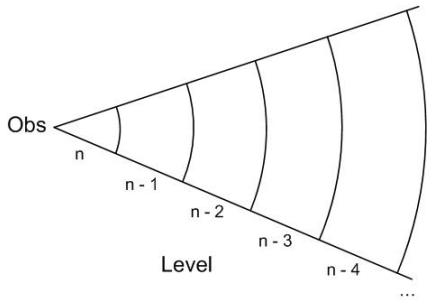


Figure 3: The level for a node is computed depending on the distance to the user.

processed by the node A8, while the relevant edges 3, 4, 5, 6 and 7 will be processed by the root node A (because even though they belong to the A8 subtree, they are not internal to it). The process for each relevant edge consists only on the generation of the triangle (T-edge) or triangles (X-edge) joining the points representing the nodes that are involved in the edge (see figure 1).

7 RESULTS AND DISCUSSION

We have implemented a prototype version of the proposed technique and we have measured its performance with several test models.

The first model is a voxel model of a jaw (see figure 4) containing a total of 1.139.816 voxels. From these, 49.998 are grey voxels (which become membrane nodes at the minimal resolution of the octree). The complete mesh generated from these terminal membrane nodes, without simplification, contains a total of 84.190 triangles.

Figure 5 shows the total number of triangles sent to OpenGL by our algorithm during a short and simple navigation. This navigation consists on moving the camera closer to the jaw and going back. We can observe that when the camera is close to the model the total number of triangles drawn is near to the complete mesh because the level-of-detail should be high enough, while when the camera is going back the number of triangles drawn decreases again.

In figure 6 we can see the number of triangles re-used on each frame during the same short navigation. We can observe that the number of triangles re-used is near to the number of triangles drawn except in those frames where the number of triangles to draw changes (compare this with figure 5 which shows the number of triangles drawn at each frame and also with figure 7 which shows the percentage of triangles re-used). This re-using of triangles already generated can be optimized by using vertex buffer objects of the GPU.

A snapshot of a second model can be seen in figure 7. This model has a total of 2.004.504 voxels with 64.577 of them being grey voxels. The complete mesh generated from these terminal membrane nodes, without simplification, contains a total of 108.534 triangles.

8 CONCLUSION AND FUTURE WORK

We have presented a new technique for a view-dependent LOD rendering, which builds an octree-based data structure enabling the extraction of arbitrary view-dependent LODs and renders the scene by using

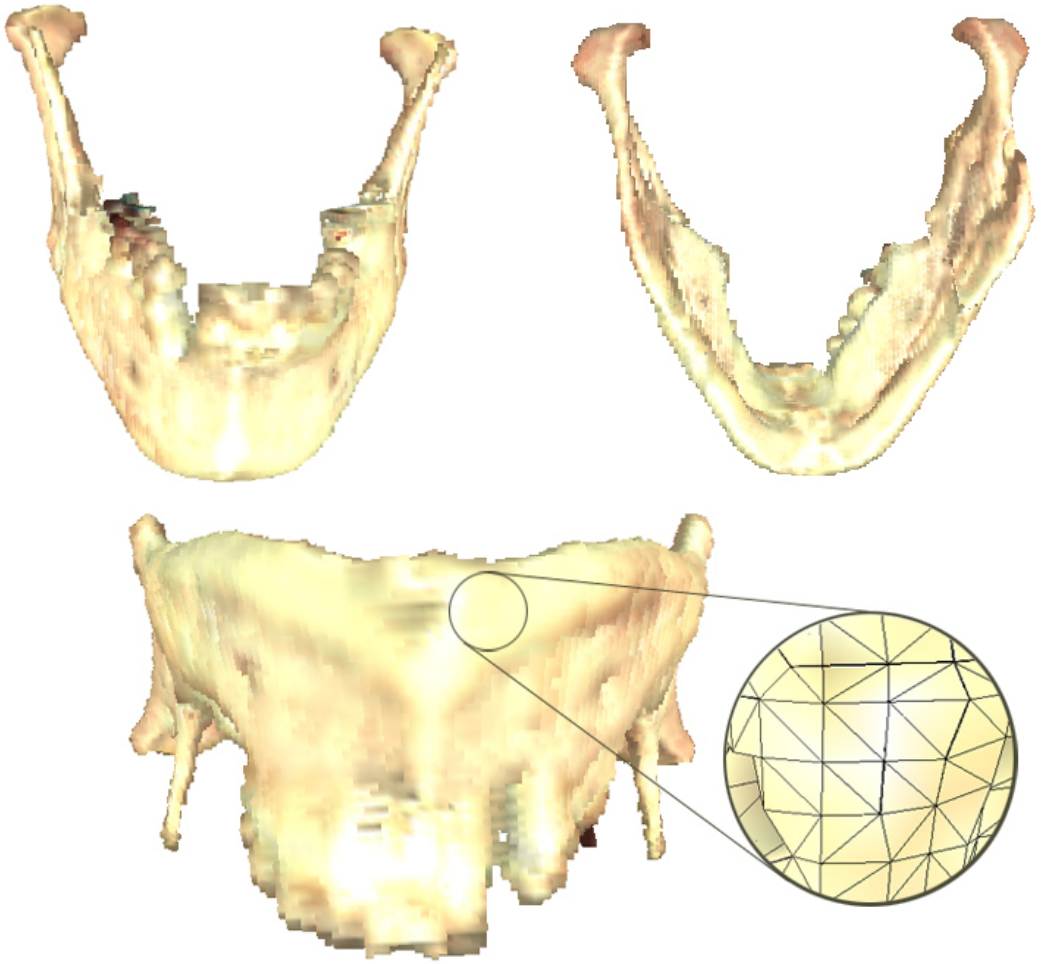


Figure 4: Several views of the model of a jaw during the navigation. The step on the left side of the magnified view is also present in the original model.

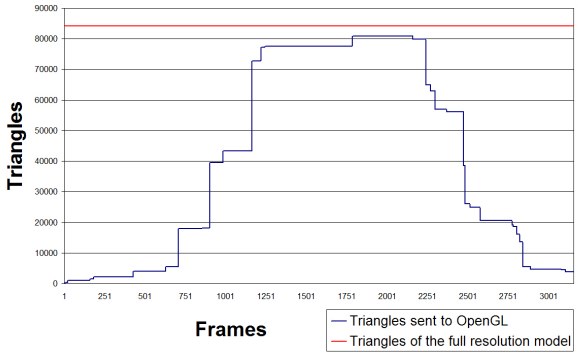


Figure 5: Triangles sent to OpenGL compared with the triangles of the full resolution model.

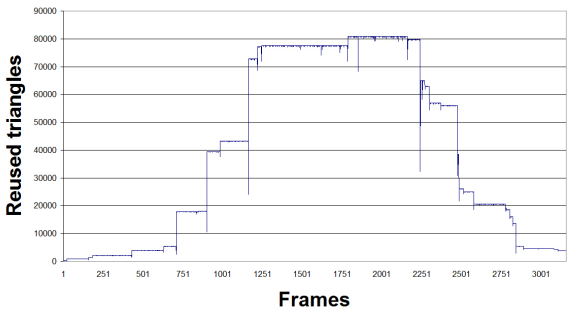


Figure 6: Total number of reused triangles per frame.

a depth-first traversal algorithm which allows to re-use triangles from one frame to another.

As a future work we can improve the efficiency of our current prototype by including the use of vertex buffer objects for the triangles to be re-used among frames and also by generating strips of triangles instead of triangles.

We will also include in the algorithm the possibility of having out-of-core data, so our octree will be stored completed in external memory while having the front nodes kept in internal memory. This will require some

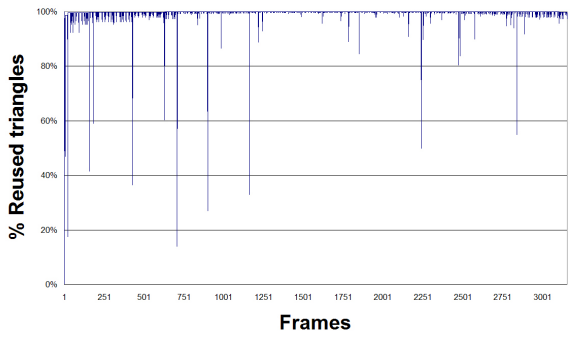


Figure 7: Percentage of reused triangles per frame.

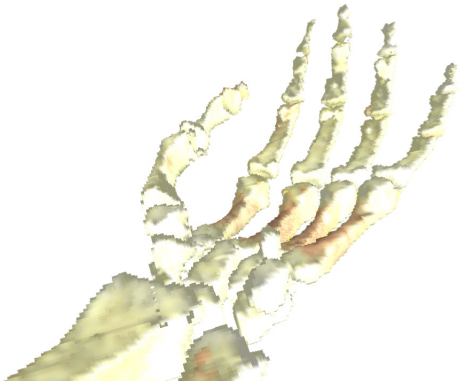


Figure 8: Snapshot of the model of a hand.

pre-fetching techniques to avoid latency on changing the nodes being kept in internal memory.

Finally, another future work is to allow the selection of objects during the navigation, which will permit the user to see an object in more detail than others, for example. To do this, it will be necessary to keep a list of object identifiers at each leaf node of the octree in order to represent correctly those nodes at the minimal resolution.

ACKNOWLEDGEMENTS

We want to thank very specially the collaboration of our colleagues Carlos Andújar, Pere Brunet, Isabel Navazo and Àlvar Vinacua for their valuous help on this work.

This work has been partially supported by the TIN2004-08065-C02-01 project of the Spanish government.

REFERENCES

[1] Pere Brunet and Isabel Navazo. Solid representation and operation using extended octrees. *ACM Transactions on Graphics*, 9(2):170–197, 1990.

[2] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics*, 23(3):796–803, 2004.

[3] Jihad El-Sana and Yi-Jen Chiang. External memory view-dependent simplification. *Computer Graphics Forum*, 19(3), 2000.

[4] Klaus Engel and Thomas Ertl. Interactive high-quality volume rendering with flexible consumer graphics hardware. In *State of The Art Report. Eurographics'02*, 2002.

[5] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346. ACM Press, 2002.

[6] Massimiliano B. Porcu and Riccardo Scateni. An iterative stripification algorithm based on dual graph operations. In *EuroGraphics 2003 (short presentations)*, pages 69–75, September 2003.

[7] Gokul Varadhan, Shankar Krishnan, T. V. N. Sriram, and Dinesh Manocha. Topology preserving surface extraction using adaptive subdivision. In *SGP'04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 241–250. ACM Press, 2004.

[8] Yucel Yemez and Francis Schmitt. Multilevel representation and transmission of real objects with progressive octree particles. *IEEE Transactions on Visualization and Computer Graphics*, 09(4):551–569, 2003.