Frame-to-Frame Coherence of Multiresolution Ordered Meshes

José Ribelles, Miguel Chover, Angeles López, Joaquín Huerta
Departamento de Informática
Universidad Jaume I. 12071. Castellón
e-mail: ribelles@uji.es

Abstract

Multiresolution models represent objects providing a wide range of resolutions or levels of detail (LOD). Efficient storage and retrieval of the different LODs is necessary in order to achieve interactive visualization. In this paper we present a data structure and algorithm that, using a Multiresolution Ordered Meshes (MOM) representation, allow us to extract uniform resolution LODs exploiting the frame-to-frame coherence property. We study advantages and drawbacks of this new approximation by testing with different data sets and comparing the results obtained with the previous MOM representation.

Keywords: multiresolution modeling, uniform resolution LOD, frame-to-frame coherence.

1 Introduction

One of the main goals in multiresolution modeling is to achieve interactive visualization of objects consisting of thousands of polygons. Multiresolution models usually store *n* different resolutions or LODs of an object and use the most suitable based on a criterion such as distance to the viewer or importance in the scene. In order to display a LOD, the visualization algorithm traverses the data structure looking for all the triangles pertaining to that particular LOD. This process must be done for each frame as quickly as possible. There is a high probability that the required LODs to be displayed in two consecutive frames will be the same or very similar. Therefore, instead of searching for all the faces in each frame, the frame-to-frame coherence can be exploited so that only the set of faces that change between the two required LODs are searched for, improving the visualization algorithm performance.

Ribelles et al. [1, 2] introduced the multiresolution model called MOM. This model has been proposed as an efficient model to manage uniform LOD but it does not exploit the frame-to-frame coherence property. In this paper, we propose a new visualization algorithm and data structure that, exploiting this property, improve the LOD retrieval process. This new approximation has been tested using different data sets in order to measure its performance compared to the previous MOM representation.

Other multiresolution models that take advantage of the frame-to-frame coherence property have been proposed in the literature. Hoppe [3] presented the Progressive Mesh (PM) representation, a new scheme for storing and transmitting arbitrary triangle meshes. This method has been extended in different ways. Xia et al. [4, 5] proposed an algorithm to perform adaptive real-time LOD based rendering; Campagna and Seidel [6] presented an alternative method to generate PM representations; Hoppe [7] introduced a framework for selective refinement and Gueziec et al. [8] described a new scheme for partitioning a surface in levels-of-detail. Finally, Hoppe [9, 10] presented data structures and algorithms for efficient implementation.

The Multi-Triangulation (MT) [11, 12] is a general framework for modeling surface meshes at variable LODs. Floriani et al. [13] presented different algorithms to construct a MT, and an algorithm for extracting a representation at variable resolution, which is interruptible. In the same year, Klein and Krämer [14] presented an algorithm as an extension of the algorithm described by Puppo [11] to extract triangulations at variable resolutions. Again Floriani et al. [15], proposed an efficient implementation of a MT representation and alternative data structures for encoding it.

The remainder of the paper is organized as follows. In section 2, the MOM representation is reviewed. In section 3, a description of the new data structures, the construction process and the visualization algorithm are presented. In section 4, we show the results obtained from tests with different data sets. Finally, conclusions and future work are discussed in section 5.

2 Review of Multiresolution Ordered Meshes

2.1 Data structures

Multiresolution Ordered Meshes is a model where data are stored explicitly. Consequently, all the vertices and triangles of the original mesh and all the vertices and triangles used in any LOD must be stored. The data structure of a MOM representation is shown in figure 1.

```
struct momVertex {
  float x, y, z;
  int firstDeletedTriangle;
};
struct momTriangle {
  struct momVertex v1, v2;
};
struct momAssociatedTriangle{
  int triangle;
};
struct momLOD {
  int deletedVertices;
  int triangles;
};

struct MOM {
  int nVertices, nTriangles;
  struct momVertex momVertexList[];
  struct momTriangle momTriangleList[];
  struct momAssociatedTriangle
         momAssociatedTriangleList[];
  struct momLOD momListControl[];
};
```

Figure 1: MOM data structure.

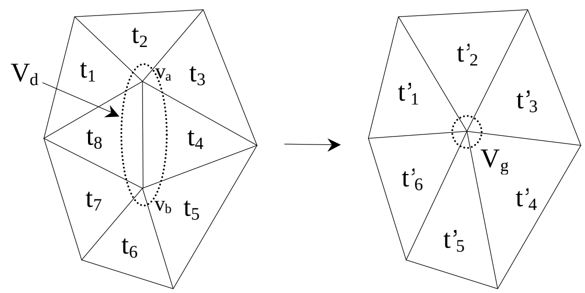
Vertices are stored in the array called *momVertexList*. Each vertex stores, beside its coordinates, a pointer (the index) to the set of triangles to be deleted when the vertex is deleted (see section 2.2). These sets of triangles (their indices) are stored consecutively in the array called *momAssociatedTriangleList*. Triangles are stored in the array called *momTriangleList*, but a MOM representation only needs two pointers per triangle, as we will explain later. For each LOD, it is necessary to store the number of vertices that do not belong to that particular LOD and the number of triangles that are possible candidates for its representation. This information is stored in the array called *momListControl* and it is used to find out the index of the first vertex that pertains to the required LOD, and to know how many triangles to be retrieved.

2.2 Construction process

In order to construct the MOM representation, we need a simplification method. Any simplification method can be used. In general, any method makes multiple passes over the mesh. During a pass, a set of vertices and the triangles that use these vertices are deleted. The resulting hole in the mesh is patched by forming a local triangulation. This process is repeated until some termination condition is met.

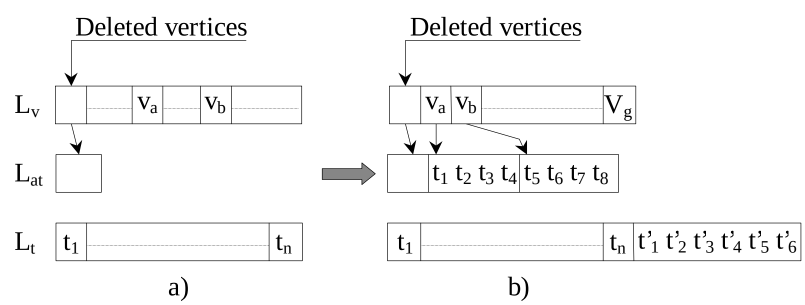
We initialize the data structure storing the original mesh. After each pass of the simplification process (figure 2), we append the changes to the data structure. These changes are the following:

- The vertices to be deleted (V_d).
- The triangles to be deleted (T_d).
- The new vertices generated (V_g).
- The new triangles generated (T_g).



$T_d = t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8$ $\qquad$ $T_g = t'_1, t'_2, t'_3, t'_4, t'_5, t'_6$

Figure 2: A pass of the simplification process.



L_v: momVertexList; L_{at}:momAssociatedTriangleList; L_t: momTriangleList

Figure 3: Changes appended to the data structure: a) before; b) after.

Each vertex v_i in V_d is moved in the *momVertexList* array, so that the first vertices to be deleted are the first vertices in the array (figure 3). Therefore, vertices are ordered while the MOM representation is constructed. The indices of the triangles to be deleted T_d are distributed among the vertices in V_d. The triangles associated with each vertex are ordered depending on their position in the *momTriangleList* array (figure 3) and they are stored at consecutive positions in the *momAssociatedTriangleList* array. Each vertex v_i in V_d stores the position of the first of its associated set of triangles.

The vertices in V_g are stored at the end of the *momVertexList* array, and the triangles in T_g are appended at the end of the *momTriangleList* array at consecutive positions. Therefore, triangles are stored in order depending on the moment they are generated. In addition, the number of vertices that do not belong to the LOD and the number of triangles that are candidates for its representation are stored in the *momListControl* array.

When the simplification process has finished, the remaining triangles are distributed among the remaining vertices in the same way as during the simplification process. After this, any triangle is pointed by a vertex and this is the reason why a MOM representation only needs two pointers (just the other two vertices) per triangle.

2.3 LOD visualization

Due to the ordering of vertices and triangles (order generated in the construction process) and the control information, the algorithm shown in figure 4 can retrieve a mesh efficiently. This algorithm traverses the *momVertexList* array and for each vertex v_i, their associated triangles are displayed if their indices are less or equal to t (value obtained from the *momListControl*).

```
/* given a LOD, from control list extract: */
vi ← the first vertex belong to that LOD
t ← #triangles candidates for representing it

/* we calculate m as a function of i and t: */
m ← #triangles pertaining to that LOD

/* while there be triangles to display */
while (m> 0)
     /* traverse the list of triangles associated to the vertex vi*/
     while (no empty triangle_list (vi)) and (triangle index ≤ t)
          display triangle
          m= m- 1
     end while
     i= i+ 1 /* proceed to the next vertex */
end while
```

Figure 4: LOD visualization algorithm.

3 Frame-to-frame coherence

The data structure and the algorithm used to, given a LOD, store and retrieve the triangles of a MOM representation are shown in the previous section. The algorithm does not exploit the frame-to-frame coherence property. In order to achieve this property, some changes in the data structure and in the algorithm must be made and they are presented in this section.

Let T_a be the set of triangles of the LOD displayed in the actual frame, and let T_b be the set of triangles of the LOD to be retrieved in the next frame. Let t_a be the triangles in T_a that do not belong to T_b, and let t_b be the triangles in T_b that do not belong to T_a. If we calculate the set $(T_a - t_a) \cup t_b$ the result is T_b. The problem we try to solve is to obtain t_a and t_b efficiently.

3.1 Data structures

We scale the data structure of the MOM representation by adding two new arrays that are shown in figure 5. The *momLastDeletedTriangleList* array stores for each vertex the position of its last triangle retrieved (figure 6). Therefore, the *firstDeletedTriangle* field in the *momVertexList* array stores the first triangle retrieved and the correspondent entry in the *momLastDeletedTriangleList* array stores the last triangle retrieved.

The *momTriangleToVertexList* array stores, for each triangle t_i in *momTriangleList* array, the position of the vertex that points to t_i (figure 6).

```
struct momLastDeletedTriangle {
  int lastDeletedTriangle;
};
struct momTriangleToVertex {
  int vertex;
};
struct momLastDeletedTriangle momLastDeletedTriangleList[];
struct momTriangleToVertex momTriangleToVertexList[];
```

Figure 5: The data structure added to MOM.

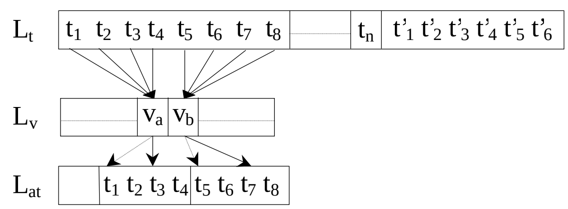


Figure 6: Now, we know the first and the last triangle retrieved per vertex, and the vertex that points to each triangle.

3.2 Construction process

The two new arrays described in the previous section, *momLastDeletedTriangleList* and *momTriangleToVertexList*, are generated in real time once the MOM representation has been read from either a disk file or a communication line. Both data structures are generated as dynamic memory structures and do not need to be stored next to the MOM representation.

The *momLastDeletedTriangleList* array is generated using the algorithm shown in figure 7. Given a LOD, the algorithm traverses the vertex list, and for each vertex, its associated triangle list is traversed and the position of the last triangle is stored in the array.

```
/* given a LOD, from control list extract: */
t ← #triangles candidates for representing the LOD

/* traverse the vertex list */
for each vertex vi
     /* traverse the list of triangles associated to the vertex vi*/
     while (no empty triangle_list (vi)) and (triangle index ≤ t)
       last_triangle= triangle index;
     end while
     momLastDeletedTriangleList[vi]= last_triangle;
end for
```

Figure 7: Generation algorithm of the *momLastDeletedTriangleList* array for the best LOD

The *momTriangleToVertexList* array is generated using the algorithm shown in figure 8. For each vertex, all of its associated triangles are traversed. Therefore, we know the vertex that points to each triangle and it is stored in the *momTriangleToVertexList* array.

```
/* traverse the vertex list */
for each vertex vi
     /* traverse the list of triangles associated to the vertex vi*/
     for each triangle tj
          momTriangleToVertexList[tj]= vi
     end for
end for
```

Figure 8: Generation algorithm of the *momTriangleToVertexList* array

3.3 LOD visualization

The main feature of this algorithm is its dependence on the LOD retrieved in the last frame. That is, the algorithm starts from the LOD displayed in the previous frame and performs a sequence of operations until reaching the required LOD in the actual frame. The two new operations are the following:

- momNext: go to the next better LOD.
- momPrev: go to the next worse LOD.

Given a LOD L_a, we want to display a new LOD L_b. If L_a is worst than L_b the algorithm performs the *momNext* operation *a-b* times in order to reach L_b. Otherwise, the algorithm would perform the *momPrev* operation *b-a* times.

```
/* from control list extract: */
t ← #triangles candidates for representing the next better LOD
tb ← the set of triangles that pertains to the next better LOD

/* traverse the set of triangles tb */
for each triangle tbi in tb
     vi= momTriangleToVertexList[tbi];
     start= momLastDeletedTriangleList[vi];
     end= momVertexList[vi].firstDeletedTriangle;
     /* trav. in falling order the list of tri. assc. to vertex vi*/
     while (start>= end) and (triangle index > t)
       last_triangle= triangle;
     end while
     momLastDeletedTriangleList[vi]= last_triangle;
end for
```

Figure 9: *momNext* operation.

The *momNext* operation extracts from the *momListControl* the set of triangles t_b that pertain to the next better LOD. This can be made easily due to the order of the *momTriangleList* array. Then, for each triangle in t_b we extract from the *momTriangleToVertexList* array the vertex and we update the *momLastDeletedTriangleList* array only for this vertex. The update is performed starting from the *lastDeletedTriangle* toward the *firstDeletedTriangle*. The algorithm is shown in figure 9.

```
/* from control list extract: */
t ← #triangles candidates for representing the next worst LOD
tb ← the set of triangles that pertains to the next worst LOD

/* traverse the set of triangles tb */
for each triangle tbi in tb
     vi= momTriangleToVertexList[tbi];
     start= momLastDeletedTriangleList[vi];
     end= momVertexList[vi+1].firstDeletedTriangle;
     /* traverse the list of triangles associated to the vertex vi*/
     while (start< end) and (triangle index ≤ t)
       last_triangle= triangle;
     end while
     momLastDeletedTriangleList[vi]= last_triangle;
end for
```

Figure 10: *momPrev* operation.

```
/* perform the sequence of operations */
if (lastLOD< requiredLOD)
    while (lastLOD< requiredLOD) { momPrev (); lastLOD++; }
  else if (lastLOD> requiredLOD)
    while (lastLOD> requiredLOD) { momNext (); lastLOD--; }

/* from control list extract: */
vi ← the first vertex belong to the requiredLOD
t ← #triangles candidates for representing it

/* we calculate m as a function of i and t: */
m ← #triangles pertaining to the requiredLOD

/* while there be triangles to display */
while (m> 0)
     start= momVertexList[vi].firstDeletedTriangle;
     end= momLastDeletedTriangleList[vi];
     /* traverse the list of triangles associated to the vertex vi*/
     for (cont= start; cont<= end; cont++)
          display triangle
          m= m- 1
     end while
     i= i+ 1 /* proceed to the next vertex */
end while
```

Figure 11: Frame-to-frame coherence algorithm.

The *momPrev* operation extracts from the *momListControl* the set of triangles t_b that pertain to the next worse LOD. The main loop is the same one that the *momNext* operation but the update is performed starting from the *lastDeletedTriangle* towards the end of the list (figure 10).

Once the necessary sequence of operations has been performed, the triangles are retrieved using the algorithm shown in figure 11. We can see that the *while* loop of the algorithm shown in figure 4 has been suppressed and in its place a *for* loop appears.

4 Results

The experiments have been performed using:

- **Hardware**: Silicon Graphics workstation Reality Engine 2, with a MIPS R10000, 194MHz and 256Mb RAM.
- **Software**: the multiresolution model has been encoded in C and OpenGL. The simplification method used in the experiments is based on edge collapses, proposed by Garland and Heckbert [16].
- **Data sets**: the meshes are from Stanford University Computer Graphics Laboratory (http://www-graphics.stanford.edu/data/3Dscanrep/) and Cyberware (http://www.cyberware.com/models/index.html). They are shown in figure 14.

Hereafter we refer to the LOD visualization algorithm (figure 4) as MOM and the LOD visualization algorithm with frame-to-frame coherence as FFC-MOM.

Figure 12 shows the performances of both algorithms with different objects. The abscise axis represents the retrieved LOD (the level 0 is the highest resolution LOD), and the ordinate axis represents the retrieval time in seconds. Four curves are drawn for FFC-MOM since the performances of this algorithm are dependent on the previous retrieved LOD. Each curve has been obtained by requesting a number of LODs (20, 50, 200 and 1000) that are related linearly, so that the difference between consecutive LODs is a constant step. We can observe that if the number of LODs increases (the step is smaller), the extraction time decreases, given that the number of operations performed (*momNext* or *momPrev*) between two consecutive displayed LODs decreases. Table 1 shows the mean times of the experimental results showed in this figure.

Figure 13 shows the extraction times of the middle LOD for both algorithms when any other LOD was displayed in the previous frame. We can observe the interval of LODs around the required LOD for which FFC-MOM behaves better than MOM and vice versa. The retrieval time for MOM is constant because it is

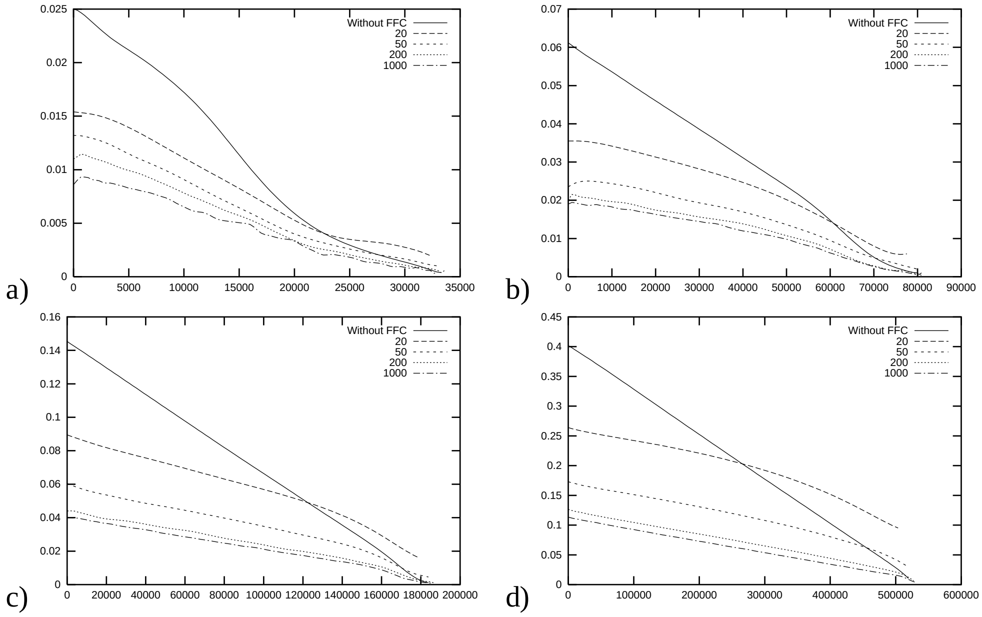


Figure 12: Performances of both algorithms. a) bunny, b) phone, c) isis, d) buda.

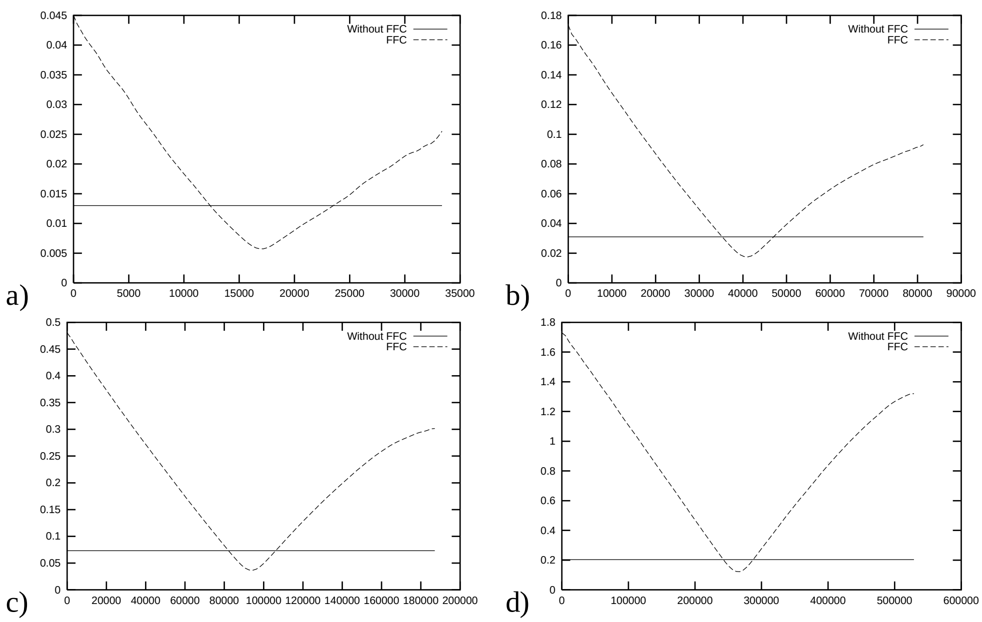


Figure 13: Performances of both algorithms when the middle LOD is extracted.
a) bunny (16,970), b) phone (40,823), c) isis (93,810), d) buda (265,066).

independent on the previous retrieved LOD. The FFC-MOM has the best performance when the required LOD is near of the LOD in the previous frame and it gets worse as the LOD of the previous frame goes away.

Table 2 shows the storage cost of the data structures of the MOM representation and the MOM representation with frame-to-frame coherence. We can observe the increment in the storage cost is near 30%.

Algorithm	#LODs	Bunny	Phone	Isis	Buda
MOM	-	0.013	0.031	0.073	0.203
FFC-MOM	20	0.009	0.025	0.064	0.217
	50	0.006	0.017	0.039	0.125
	200	0.005	0.014	0.027	0.080
	1000	0.004	0.012	0.024	0.067

Table 1: Retrieval mean times in seconds.

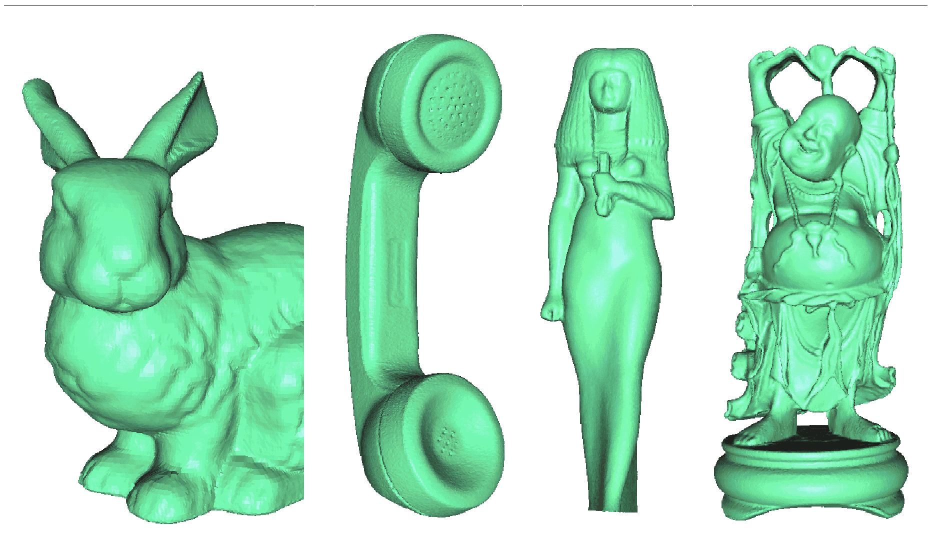
Attribute	Bunny	Phone	Isis	Buda
Vertices	34,834	83,044	187,870	543,652
Faces	69,451	165,963	375,736	1,085,636
Mb.	1.222	2.918	6.445	18.646
LODs	33,942	81,647	187,620	530,133
Mb. MOM	3.028	7.285	16.751	46.061
Mb. FFC-MOM	3.883	9.343	21.619	59.375
Mb Increase %	28%	28%	29%	29%

Table 2: Storage Costs.

5 Conclusions and future work

In this paper we present an algorithm to visualize level of details with uniform resolution exploiting the frame-to-frame coherence property of MOM representations. The data structure has been scaled in order to afford the new algorithm. New data need not be stored alongside the MOM representation, since they are generated in real time once it has been read. However, an increment in the storage cost in memory is present and the experimental results show that it is less than 30%. The experimental results show the advantages and drawbacks of the new algorithm regarding the difference between the displayed LODs in consecutive frames and their resolution level. The frame-to-frame coherence algorithm is beneficial when the difference between LODs in consecutive frames is small and the required resolution is high.

Some important issues that remain to be studied in more detail are adapting the MOM representation to variable resolution, encoding the model efficiently to improve the LOD retrieval time and applying compression techniques to decrease the storage cost.



a b c d

Figure 14: Objects used in the experiments: a) bunny, b) phone, c) isis, d) buda.

References

1. Ribelles, J., Chover, M., Huerta, J. and Quirós, R., "Un Modelo Multirresolución para Visualización Interactiva", Proc. of *CEIG'98*, Ourense, June 1998, pp. 267-280.

2. Ribelles, J., Chover, M., Huerta, J. and Quirós, R., "Multiresolution Ordered Meshes", Proc. of *1998 IEEE Conference on Information Visualization*, London, July 1998, pp. 198-204.

3. Hoppe, H., "Progressive Meshes", Proc. of *SIGGRAPH '96*, 1996, pp 99-108.

4. Xia, J. and Varshney, A., "Dynamic view-dependent simplification for polygonal models", Proc of *IEEE Visualization'96*, Ed. R. Yagel, G. Nielson, 1996, pp. 327-334.

5. Xia, J., El-Sana, J. and Varshney, A., “Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models”, Proc of. *IEEE Transactions on Visualizations and Computer Graphics*, 1997, 3, 2, pp.171-183.

6. Campagna, S. And Seidel, H., “Generating and Displaying Progressive Meshes”, in H.-P. Seidel, B. Girod, H. Niemann (eds.), *3D Image Analysis and Synthesis '97*, pp. 35-42, infix, 1997

7. Hoppe, H., “View-Dependent Refinement of Progresive Meshes, Proc. of *SIGGRAPH'97*, Aug., 1997, pp 189-198.

8. Guéziec, A., Taubin, G., Lazarus, F. and Horn, W., “Simplicial maps for progressive transmission of polygonal surfaces”, Proc. of *the third symposium on Virtual reality modeling language*, 1998, pp 25 - 31.

9. Hoppe, H., “Efficient Implementation of Progressive Meshes”, Computer & Graphics, Vol. 22, No 1, pp 27-36, 1998.

10. Hoppe, H., “Smooth view-dependent level-of-detail control and its application to terrain rendering”, Proc. of *IEEE Visualization '98*, pp 35-42;516, 1998.

11. Pupo, E., “Variable Resolution Terrain Surfaces”, Proc. of *Eight Canadian Conference on Computational Geometry*, pp. 202-210, 1996.

12. De Floriani, L., Puppo, E. and Magillo, P., “A Formal Approach to Multiresolution Hupersurface Modeling”, *Geometric Modeling: Theory and Practice*, W. Straber, R. Kein, R. Rau (Editors), Springer-Verlag, 1997.

13. De Floriani, L., Magillo, P. and Puppo, E., “Building and Traversing a Surface at Variable Resolution”, Proc. of *IEEE Visualization'97*, pp.103-110, 1997.

14. Klein, R. and Krämer, J., “Building multiresolution models for fast interactive visualization”, Proc. of *SCCG'97*, 1997.

15. De Floriani, L., Magillo, P. and Puppo, E., “Efficient Implementation of Multi-Triangulations”, Proc. of *IEEE Visualization'98*, 1998.

16. Garland, M. and Heckbert, P., "Surface Simplification Using Quadric Error Metric", Proc. of *SIGGRAPH'97*, pp. 209-216, Aug., 1997.