Informe Técnico
DI 01-07/99

A First Step to Evaluate and Compare Multiresolution Models

José Ribelles, Miguel Chover, Angeles López, Joaquín Huerta

Departamento de Informática

Correo electrónico: `ribelles@uji.es`

Universidad Jaime I
U. P. Informática
Campus de Riu Sec, s/n
E-12071
Castellón
España

A First Step to Evaluate and Compare Multiresolution Models

José Ribelles, Miguel Chover, Angeles López, Joaquín Huerta
Dpto. de Informática, Universitat Jaume I, 12071, Castellón, Spain
E-mail:ribelles@uji.es

Abstract:

A variety of multiresolution models for arbitrary triangle meshes have been presented in the last years. In order to achieve an interactive visualization, multiresolution models must store and retrieve data efficiently. In this paper, we present a first step to evaluate and compare multiresolution models. We propose a set of tests to measure the speed performance of the models for interactive visualization and uniform resolution LOD. We have carried out some experiments with implementations of the following two models: Progressive Meshes and Multiresolution Ordered Meshes.

Keywords:

Multiresolution modeling, level of detail, interactive visualization, uniform resolution, triangle meshes.

Un primer paso para Evaluar y Comparar Modelos Multirresolución

José Ribelles, Miguel Chover, Angeles López, Joaquín Huerta

Dpto. de Informática, Universitat Jaume I, 12071, Castellón, Spain

E-mail:ribelles@uji.es

Resumen:

Una amplia variedad de modelos multirresolución para mallas arbitrarias de triángulos han sido presentados en los últimos años. Con el objetivo puesto en conseguir una visualización interactiva, los modelos multirresolución han de almacenar y recuperar los datos eficientemente. En este artículo, se presenta un primer paso para evaluar y comparar los distintos modelos multirresolución. Se proponen un conjunto de tests que intentan medir las prestaciones de los modelos cuando son utilizados para visualización interactiva con nivel de detalle constante. Se han realizado experimentos con las implementaciones de los dos siguientes modelos: Progressive Meshes y Multiresolution Ordered Meshes.

Palabras clave:

Modelado multirresolución, nivel de detalle, visualización interactiva, resolución constante, mallas de triángulos.

A First Step to Evaluate and Compare Multiresolution Models

J. Ribelles, M. Chover, A. López and J. Huerta

Dpto. de Informática, Universitat Jaume I, 12071, Castellón, Spain

1. Introduction

Interactive applications have increased enormously their use of highly detailed objects in the last few years. These objects are often represented as polygonal meshes for efficient display and the resulting meshes involve an important increase of storage and transmission costs, and make more difficult, if not impossible, interactive visualization. The use of simplification methods[20] is a solution to this problem that achieves a compact approximation of the original model with a smaller set of polygons and guaranteed approximation error, reducing disk and memory requirements and speeding network transmission. To further improve rendering performance, it is common to define various LODs of the object. The coarsest approximations are used when the viewer is far from the object, while higher detail versions are substituted as the viewer approaches. However, visual discontinuities are encountered when switching between two LODs.

Multiresolution modeling[10] tries to solve this problem by maintaining multiple LODs. It has been proposed as an efficient method for representing complex meshes and specially useful for interactive visualization, progressive transmission, geometry compression, LOD approximation and selective refinement. Puppo and Scopigno[20] define the term "multiresolution mesh" as follows:

> Informally, a multiresolution mesh is a model that can provide a high number (virtually, a continuous range) of meshes representing a single object at different resolutions. The number of different levels of detail that such a model can provide will not be fixed a priori, but it will rather be proportional to the size of the mesh representing the object at the maximum possible resolution.

Consequently, a multiresolution model must define an adequate frame for, on the one hand, storing the different resolutions and, on the other hand, managing them. Solving with more or less success the problems indicated above will depend on the design of the model, which directly influences its performance. Thus, we might find a model with better performance for transmission, or with a higher level of compression, or more agility when displaying an approximation of the original object. There are also other features that determine the use of a model, such as its dependence on the simplification method, or its capacity to represent any type of surface.

In this paper, the first step towards the evaluation and comparison of multiresolution models is presented. Until now, studies with this objective have not been published. It is very difficult to make a general evaluation of models due to the different purposes for which they have been developed. For this reason, we have focused our initial work on *interactive visualization* with *uniform resolution LOD*. For this purpose, a set of tests has been defined consisting of criteria to measure the speed performance of a model in such a way that the obtained results can serve as a reference. We propose the following tests:

- Random: measures the performance of the model when a random sequence of LODs are required.
- Linear: measures the performance of the model when a linear sequence of LODs are required.
- Exponential: measures the performance of the model when an exponential sequence of LODs is required.

It is also interesting to study the memory usage of the model and try to establish some kind of relationship between the storage cost and speed requirements. Although there is not a specific test, the storage costs of the models are attached in the section of experimental results. For the experiments, two multiresolution models have been implemented: Progressive Meshes, introduced by Hoppe[11], and Multiresolution Ordered Meshes, introduced by Ribelles et al.[21]. The first model has been selected as an example of an implicit model (in our implementation only has been encoded the geometry[13]) and, in addition, it is an available feature of Microsoft DirectX 5.0. The other one is an explicit model. Conclusions can be reached about concrete models by the use of tests.

The rest of this paper is organized as follows. In section 2, we summarize previous work on multiresolution modeling of triangular meshes. In section 3, we describe the two implemented models, detailing data structures and algorithms. A description of the tests is presented in section 4. In section 5, we show the results obtained from the proposed tests with the implemented models. Finally, conclusions and future work are discussed in section 6.

2. Previous Work

The concept of Multiresolution Analysis (MRA) is generalized[17] for surfaces of arbitrary topological type. Eck et al.[4] present a method for overcoming the subdivision connectivity restriction, meaning that completely arbitrary meshes can be converted to multiresolution form. Later, Certain et al.[2] extend MRA to perform multiresolution analysis of colored meshes by separately analyzing shape and color.

Hoppe[11] presents 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.[23, 24] present an algorithm to perform adaptive real-time level-of-detail based rendering; Campagna and Seidel[1] present an alternative method to generate PM representations; Hoppe[12] introduces a framework for selectively refining and Gueziec et al.[9] describe a new scheme for partitioning a surface in levels-of-detail. Finally, Hoppe[13, 14] presents data structures and algorithms for efficient implementation.

The Multi-Triangulation[19, 5] is a general framework for modeling surface meshes at variable level of detail. Floriani et al.[6] present different algorithms to construct an MT, and an algorithm to extract a representation at variable resolution, which is interruptible. In the same year, Klein and Krämer[16] present an algorithm as an extension of the algorithm described by Puppo[19] to extract triangulations at variable resolutions. Again Floriani et al.[7], describe an efficient implementation of a Multi-Triangulation and alternative data structures for encoding it.

Ciampalini et al.[3] present a general multiresolution data scheme. The data structure is based on an interval tree, where each triangle stored has two time stamps. Hierarchical dynamic simplification[18] provides a framework for the dynamic view-dependent simplification, working by clustering vertices together in a hierarchical fashion. Multiresolution Ordered Meshes[21], is a new model where meshes are stored in order depending on the simplification process, and uniform level of detail retrieval can be performed efficiently.

Other multiresolution models that use binary space partition trees have been presented[22, 15]. These models take advantage of their own data structure in order to increase the performance of the management and visualization operations.

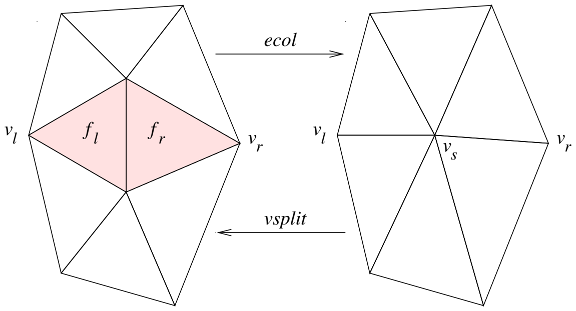


Figure 1: Vsplit *and* ecol *operations*

3. Review of the implemented multiresolution models

In this section, we review the data structures and algorithms used in the implementations of Progressive Meshes and Multiresolution Ordered Meshes, together with a brief explanation of both models, explaining the differences with the published original works. Also, the storage usage study is presented.

3.1. Progressive Meshes

This multiresolution model stores a mesh M as a coarse base mesh M^0 together with a sequence of n detail records that indicate how to incrementally refine M^0 back into the original mesh M. The simplification process for constructing a progressive mesh (PM) representation must perform edge collapses. Each record stores the necessary information to modify the mesh by introducing one new vertex. Using Hoppe's notation, this operation is called *vsplit*, and the inverse *ecol* (figure 1). Thus, the PM representation of a mesh M defines a continuous sequence of meshes $M^0, M^1, ..., M^n = M$, of increasing accuracy, from which LOD approximations of any desired complexity can be efficiently retrieved.

3.1.1. Basic Data Structures

Although the original work[11] was introduced in 1996, data structures and algorithms for efficient implementation of the PM scheme were presented[13] in 1998. In this last work, Hoppe offers solutions to encode appearance attributes of a mesh including as discrete attributes (material identifier) and as scalar attributes (normals and texture coordinates). We have excluded this information in our implementation, encoding only the geometry.

In order to construct a PM representation a data structure must be defined to represent the mesh (figure 2), and another data structure to represent the sequence of records (figure 3). As shown in figure 2, a mesh is defined to contain an array of vertices and an array of faces where, besides pointing to vertices each face stores pointers to its three neighbouring faces (-1 if neighbour does not exist). This is necessary in order to perform *vsplit* and *ecol* operations efficiently.

```
struct Vertex { /* x,y,z coordinates */
  float coord[3];
};
struct Face {
  int vertex[3];/* pointers to vertices */
  int fnei[3];  /* p. to face neighbors */
};
struct Mesh {
  struct Vertex vertices[];
  struct Face faces[];
};
```

Figure 2: *The Mesh data structure*

```
struct PMesh {
  struct Mesh baseMesh;  /* base mesh M^0 */
  struct Vsplit vsplits[]; /*array of vertex
                              split records */
  int fullNvertices;/* # vertices in M */
  int fullNfaces;   /* # faces in M */
};
```

Figure 3: *The PM data structure*

As shown in figure 3, a PM representation is defined to contain a base mesh (*baseMesh*) to store M^0, using the structure shown in the figure 2, and an array of vertex split records (*vsplits*). Information about the original mesh, M, is also included for efficient pre-allocation of arrays.

Encoding of a vsplit operation is shown in figure 4. We identify the location of the vertex split within the mesh through the index of a face *flclw* (figure 5). The v_s vertex is specified as an index (*vsIndex*) in the range $[0..2]$ into the ordered vertices of face *flclw*. We store the number of clockwise face rotations (*vlrRot*) about v_s from v_l until v_r, to determine the other vertex v_r. The face adjacency information (*fnei*) is used to perform these rotations. Two special symbols for *vlrRot* are used for the cases when v_r or *flclw* do not exist (figure 6).

Two vertex position deltas, *vadS* and *vadL*, are stored to let the vertex split transformation modify vertex positions as

```
struct Vsplit {
  int flclw; /* a face neighbour of vsplit */
  short vlrRot; /* encoding of v_r */
  struct {      /* index of v_s in flclw */
    short vsIndex;
  } code;
  float vadL[3], vadS[3];   /* deltas */
};
```

Figure 4: *Vertex split data structure*

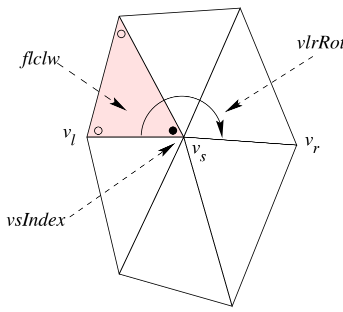


Figure 5: *The* Vsplit *parameters*

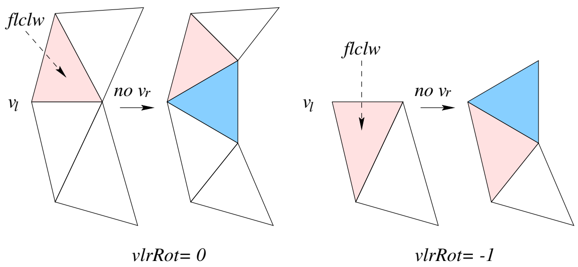


Figure 6: *Special cases*

follows:

$$vsplit \rightarrow v_t = v_s + vadS; v_s = v_s + vadL;$$
$$ecol \rightarrow v_s = v_s - vadL;$$

3.1.2. Storage Cost

This analysis is done without compression techniques and including all the fields necessary for interactive visualization (different if PM representation is used for rendering static models). All coordinates are represented as 32-bit floating-point numbers, integers are 32-bit, and shorts are 16-bit.

Let V and F be the number of vertices and faces of the original mesh M respectively. Let R be the number of records vsplit to transform the base mesh M^0 into M. We need to store all the vertices and faces of the mesh M, and we need a memory-resident *PMesh* for changing interactively the LOD. The cost of the mesh is $12V$ bytes plus $24F$ bytes. The array of *vsplit* records has enclosed its length by the number of vertices of the mesh M. The field *vsIndex* needs only two bits per record. So, the higher cost is $30.25V$ bytes. If we add all, we obtain a total cost of $24F + 42.25V$ bytes.

3.1.3. LOD visualization

In order to visualize a level-of-detail, LOD_i, after displaying a level-of-detail, LOD_j, it is necessary to apply to the mesh M^j the sequence of records from j to i, and afterwards traverse the list of faces of the actual mesh. Two functions are defined: *pmNext()* to perform a *vsplit* operation; *pmPrev()* to perform an *ecol* operation.

- **Vsplit transformation**. This appends the vertex v_t and the faces f_l and f_r (or only f_l depending on *vlrRot*) to the actual mesh. It updates the faces and the local adjacencies (*fnei*) that use v_t. Finally, it updates the vertex v_s using the deltas stored in *Vsplit*.
- **Ecol transformation**. This updates the vertex v_s. It updates the faces and the local adjacencies (*fnei*) that use v_t. Finally, it removes the vertex v_t and the faces f_l and f_r (or only f_l depending on *vlrRot*).

The LOD visualization algorithm needs to perform *n vsplit* or *ecol* operations in order to transform the actual mesh to the required mesh. Each operation appends or removes a vertex of the actual mesh. The higher number of operations is enclosed by the number of vertices in the mesh M. The worst case is to display the mesh M, when M^0 is the actual mesh (or vice versa). Therefore, all the register sequence has to be performed. Later, we need to traverse the list of faces to display the actual mesh.

3.2. Multiresolution Ordered Meshes

Multiresolution Ordered Meshes (MOM) is a model where data are stored explicitly. This model is based on an ordered sequence of the successive approximations. The simplification process produces this order, and any simplification method can be used to construct a MOM representation.

3.2.1. Basic Data Structures

To construct a MOM representation[21], a data structure must be defined (figure 7) to represent the original mesh and all the level of detail approximations. This structure stores all the vertices and triangles of the original mesh, and all the vertices and triangles generated during the simplification process. We start storing the original mesh and at each iteration of the simplification process (figure 8), we have to append the changes to the data structure. They 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.

Vertices are stored in the array called *momListVertices*. Each vertex belonging to V_d is moved in its array in such a way that the first vertices to be deleted are the first vertices in the array. Each one of these vertices store, beside coordinates, a pointer to an index of a triangle that belongs to T_d, and the triangles to be deleted T_d are stored (their indices) at consecutive positions in the *momListAssociatedTriangles* array. Triangles common to more than one of these deleted vertices appear just once. Finally, the new vertices, V_g, are stored at the end of the *momListVertices* array. Triangles are

```
struct momVertex {
  float x, y, z;
  int firstTriangleDeleted;
};
struct momTriangle {
  struct momVertex *v1, *v2;
};
struct momAssociatedTriangle {
  int triangle;
};
struct momLOD {
  int verticesDeleted;
  int triangles;
};
struct MOM {
  int nVertices, nTriangles;
  struct momVertex momListVertices[];
  struct momTriangle momListTriangles[];
  struct momAssociatedTriangle
         momListAssociatedTriangles[];
  struct momLOD momListControl[];
};
```

Figure 7: *MOM data structure*

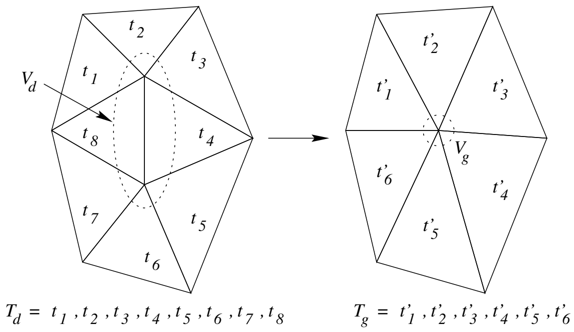


Figure 8: *Simplification process*

stored in the array called *momListTriangles*. The new triangles, T_g, are appended at the end of the list at consecutive positions, and they stay there during all the simplification process. A particularity of MOM is that only two indices define a triangle, instead of three. This is due to the LOD visualization algorithm shown later.

For each level of detail, the MOM representation needs to store how many vertices do not belong to it and how many triangles are candidates for representing it. This information is stored in *momListControl* array and it is used to know the index of the first vertex that pertain to the required LOD, and to know how many triangles there are to retrieval.

```
/* Given a LOD, from control list extract */
i → the first vertex belong to that LOD
t → # triangles candidates for representing it
/* 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 list of triangles associated to
     vertex i */
  while (not empty trianglelist(i)) and
        (triangle index ≤ t)
    display triangle
    m = m-1;
  end while
  i = i+1;    /* proceed with the next vertex */
end while
```

Figure 9: *Extraction algorithm of a given LOD*

3.2.2. Storage Cost

This analysis is done without compression techniques. All coordinates are represented as 32-bit floating-point numbers and integers are 32-bit.

Let V and F be the number of vertices and faces of the original mesh. Let V' and F' the number of vertices and faces generated by the simplification process. Let be N the number of levels of detail. The *momListVertices* array has a cost of $16(V+V')$ bytes. The *momListTriangles* array has a cost of $8(F+F')$ bytes. We have to add the cost of the *momListAssociatedTriangles* array, $4(F+F')$ bytes, and the cost of the *momListControl* array, $8N$ bytes. If we add all, we obtain a total cost of $16(V+V')+12(F+F')+8N$ bytes.

However, we can not estimate in advance the values of V', F' and N because they are dependent of the simplification method. For example, if the simplification process does not generate new vertices, and each vertex has a mean of six faces surrounding it, we obtain an approximated cost of $24V+36F$ bytes.

3.2.3. 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 9 can retrieve a mesh efficiently.

4. Tests

The proposed tests are designed to measure the speed performance of the evaluated model. Each test performs a number of requests of visualization with different levels of detail and they differ in the way the sequence is generated. If a model consists of n LODs, each test i generates a sequence of m_i LODs, $m_i < n$, with the following criteria.

4.1. Random test

This test generates a random sequence of LODs. The aim is to measure the performance of a model when consecutive requests consist of unrelated LODs, thus allowing sharp changes between consecutively requested LODs. The following code generates the random sequence:

srand (*seed*);
for i=1..m_r **do**
  LOD[i] = f(**rand**());
end for

where *seed* is a positive integer constant, and f is a function that normalizes the random number to the range $[1..n]$. This code generates the same random sequence given a constant *seed*. Therefore, it is possible to test different models with the same conditions. In order to make the test independent of the length of the sequence, the mean of the time, $\bar{t}$, measured from experiments should be considered. Thus, the total time, t_r, is divided by the total number of LODs tested, m_r, provided that m_r is large enough,

$$M_r = \frac{t_r}{m_r}$$

4.2. Linear test

The linear test consists of a linear sequence of LODs. The aim is to measure the performance of the model when LODs requested consecutively are related linearly. The difference between consecutive LODs will be a constant step, proportional to the total number of LODs.

We must take into account two important features of this test: the increasing and decreasing order and the amount of the step between LODs that are consecutive in the sequence. The following code generates an increasing linear sequence at a given step k:

$m_l = n/k$;
for i=1..m_l **do**
  LOD[i] = $i*k$;
end for

The order of the sequence could lead to different test results, due to the fact that some models do not have the same behaviour when refining than when decimating. Therefore, two kind of tests are necessary: the increasing order test (traverse the sequence forwards) and the decreasing order test (traverse the sequence backwards). The final result will be the mean of both tests.

The step value is also significant, therefore we propose a set of steps computed as a percentage of the number of LODs of the model,

$$k = 0.1\%n, 0.2\%n, 0.5\%n, 1\%n, 2\%n, 5\%n$$

in order to obtain test sets of length m_l = *1000, 500, 200, 100, 50* and *20*, respectively. Despite the fact that interactive applications usually require a display of 25 frames per second, the proposed percentages are designed to evaluate the time necessary to visualize the model under different conditions.

4.3. Exponential test

The exponential test consists of an exponential sequence of LODs. The aim is to measure the performance of the model when LODs requested consecutively are related exponentially. The curvature of the exponential relationship between consecutive LODs will be designed in such a way that changes between LODs of lower resolution are smaller and changes between LODs of higher resolution are greater. The length of the sequence, m_e, determines the steps between consecutive requested LODs. Analogously to the linear tests, we propose tests with both the increasing and decreasing order, and a set of values, m_e= *1000, 500, 200, 100, 50* and *20*, in order to test the models under different conditions. The following code generates an increasing exponential sequence of a given length m_e (figure 10):

$x = 0$;
for i=1..m_e **do**
  LOD[i] = $f(\frac{(1+C)^x-1}{C})$;
  $x = x + \frac{1}{m_e-1}$;
end for

where C is a constant such that approximately 80% of the resulting LODs are in the set of $n/2$ LODs of lower resolution and 20% are in the set of $n/2$ LODs of higher resolution, that is, $C = 20$, and f is a linear transformation of the range [0..1] of the fraction into the index range of LODs [1..n], such that 0 corresponds to the lowest resolution LOD and 1 corresponds to the highest resolution LOD. The increasing and decreasing order tests consist of traversing the four proposed sequences forwards and backwards, respectively.

5. 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 models have been encoded in C, and OpenGL. These implementations have been optimized until performance published by the authors has been obtained. The simplification method used in the experiments is based on edge collapses, proposed by Garland and Heckbert[8].

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`). Figures 11, 12, 13 and 14 show the meshes selected for the experiments.

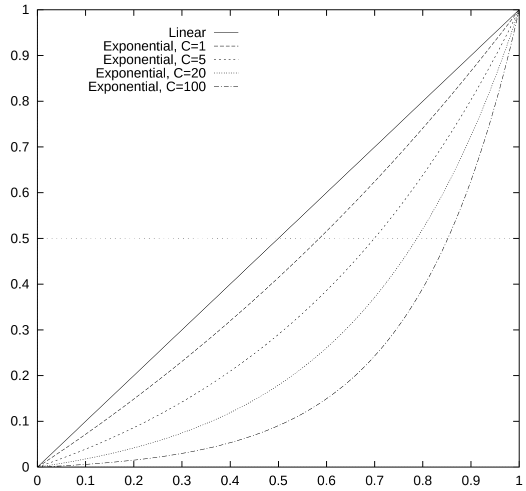


Figure 10: *Exponential sequence in range* [0..1]

Figures 11, 12, 13 and 14 show the results obtained for each mesh. Each figure consists of:

a) A view of the mesh.

b) A table that shows the mean retrieval time (in miliseconds) for each implemented model (MOM and PM), for different number of LODs (20, 50, 100, 200, 500 and 1000) and different sequence types (random, linear and exponential).

c) Random test. The abcise axis represents the LOD in a range [0,1] where 0 represents the worst LOD of the mesh and 1 the best LOD. The ordinate axis represents the retrieval time (ms.). As all the experiments have the same behaviour, we only show the results for m_r= 20, 100.

d) Linear test. The axis represent the same data than in c). MOM has the same behaviour for the different numbers of LODs and is represented with only one curve.

e) Exponential test. The axis represent the same data than in c). MOM has the same behaviour for the different numbers of LODs and is represented with only one curve.

f) This plot represents the retrieval time when the required LOD is just in the middle and the last retrieved LOD could be any.

The random test shows that MOM is more efficient when abrupt changes in the sequence of LODs are required. As the complexity of the model increases, the difference in the results becomes higher. Finally, we can observe that the random test is independent of the length of the sequence.

By comparing the random and linear tests, we can observe that the retrieval time of a LOD in MOM is independent of the previously displayed LOD. It behaves constantly with respect to the length of the sequence while PM is better as the difference between consecutive LODs is smaller. Moreover, for the linear test the difference between mean retrieval times of the two models is smaller than in the random test.

By comparing the linear and exponential tests, we can observe that the mean retrieval times have been reduced considerably, given that a higher number of LODs of lower resolution are retrieved. The difference is more significant in the case of MOM, where the exponential results become half the linear results. Again, the test shows that PM is more efficient when the difference between LODs is smaller.

Table 1 shows that the storage cost of the two models is very similar. Therefore, we cannot establish a relationship between storage cost and speed efficiency.

Attribute	Cow	Sphere	Bunny	Phone
Vertices	2,904	15,314	34,834	83,044
Faces	5,804	30,624	69,451	165,963
Mb.	0.102	0.538	1.222	2.918
LODs	2,807	15,263	33,942	81,647
Mb PM	0.255	1.349	3.065	7.316
Mb MOM	0.251	1.364	3.028	7.285

Table 1: *Storage costs*

6. Conclusions

A first step to evaluate and compare multiresolution models has been presented. The proposed tests evaluate the speed performance of the models for interactive visualization and uniform resolution LOD. Experiments with two models have been performed in order to validate their significance. The tests evidence the features of each multiresolution model.

Future work will aim to improve and make more tests to measure, for example, the relationship between storage cost and speed efficiency. One important issue that remains to be studied in more detail is adapting the tests to variable resolution LODs and focusing the tests from other points of view like progressive transmission or storage compression. Finally, we intend to make a comparison including a larger set of multiresolution models, including not only geometry but appearance attributes.

References

1. S. Campagna and H.P. Seidel. 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

2. A. Certain, J. Popovic, T. DeRose, T. Duchamp, D. Salesin and W. Stuetzle. Interactive Multiresolution Surface Viewing, *Proc. of SIGGRAPH '96*, 91–98, August, 1996.

3. A. Ciampalini, P. Cignoni, C. Montani and R. Scopigno. Multiresolution Decimation Based on Global Error, *The Visual Computer*, (13):228–246, 1997.

4. M. Eck , T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery and W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes, *Proc. of SIGGRAPH '95*, 173–182, August, 1995.

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

6. L. De Floriani, P. Magillo and E. Puppo. Building and Traversing a Surface at Variable Resolution, *Proceedings IEEE Visualization '97*, 103–110, 1997.

7. L. De Floriani, P. Magillo and E. Puppo. Efficient Implementation of Multi-Triangulations. *Proceedings IEEE Visualization '98*, 1998.

8. M. Garland and P. Heckbert. Surface Simplification Using Quadric Error Metric, *Proc. of SIGGRAPH '97*, 209–216, August, 1997.

9. A. Guéziec, G. Taubin, F. Lazarus and W. Horn. Simplicial maps for progressive transmission of polygonal surfaces, *Proc. of the 3rd. symposium on Virtual reality modeling language*, 25–31, 1998.

10. P. Heckbert and M. Garland. Multiresolution Modeling for Fast Rendering, *Proc. of Graphics Interface '94*, Banff Alberta Canada, 43–50, May, 1994.

11. H. Hoppe. Progressive Meshes, *Proc. of SIGGRAPH '96*, 99–108, 1996.

12. H. Hoppe. View-Dependent Refinement of Progresive Meshes, *Proc. of SIGGRAPH '97*, 189–198, August, 1997.

13. H. Hoppe. Efficient Implementation of Progressive Meshes, *Computer & Graphics*, 22(1):27–36, 1998.

14. H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering, *IEEE Visualization '98*, 35–42, October, 1998.

15. J. Huerta, M. Chover, J. Ribelles and R. Quirós. Multiresolution Modeling Using Binary Space Partitioning Trees, *Computer Networks & ISDN Systems*, (30:)1941–1950, 1998.

16. R. Klein and J. Krämer. Building multiresolution models for fast interactive visualization, *SCCG '97*, 1997.

17. M. Lounsbery, T. DeRose and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type, *ACM Transactions on Graphics*, 16(1):34–73, 1994.

18. D. Luebke and C. Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments, *Proc. of SIGGRAPH '97*, 199–208, August, 1997.

19. E. Puppo. Variable Resolution Terrain Surfaces, *Proc. Eight Canadian Conference on Computational Geometry*, 202–210, 1996.

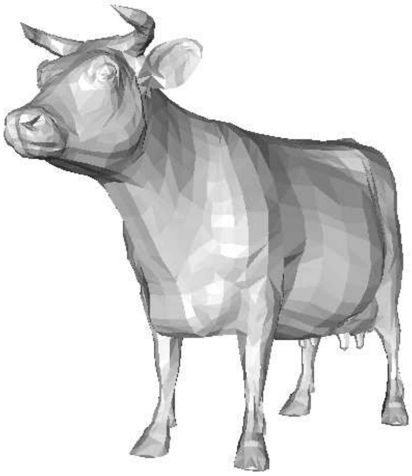
20. E. Puppo and R. Scopigno. Simplification, LOD and Multiresolution - Principles and Applications, *Eurographics'97 Tutorial Notes*, 16(3), 1997.

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

22. C. Wiley, A.T. Campbell, S. Szygenda, D. Fussell and F. Hudson. Multiresolution BSP Trees Applied to Terrain, Transparency, and General Objects, *Graphics Interface*, 88–96, May, 1997.

23. J. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models, *IEEE Visualization'96 Proceedings*, Ed. R. Yagel, G. Nielson, 327–334, 1996.

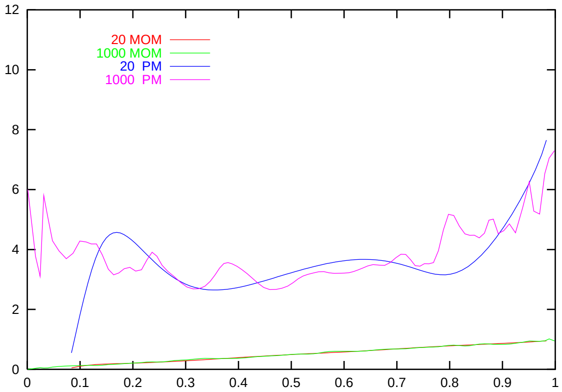
24. J. Xia, J. El-Sana and A. Varshney. Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models, *IEEE Transactions on Visualizations and Computer Graphics*, 3(2):171–183, 1997.



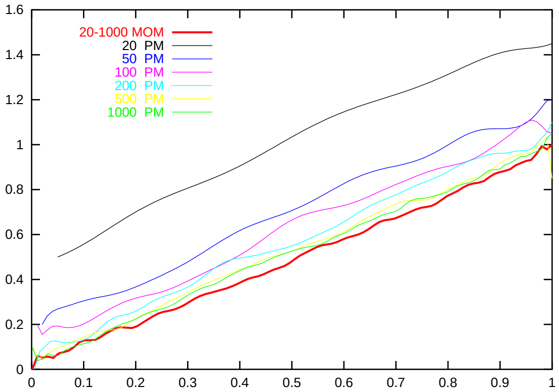
(a) Cow

#LODs	Random		Linear		Exponential	
	Mom	Pm	Mom	Pm	Mom	Pm
20	0.5	3.5	0.5	1.0	0.3	0.9
50	0.5	3.5	0.5	0.7	0.3	0.5
100	0.5	3.8	0.5	0.6	0.3	0.4
200	0.5	3.7	0.5	0.6	0.3	0.4
500	0.5	3.7	0.5	0.5	0.3	0.3
1000	0.5	3.8	0.5	0.5	0.3	0.3

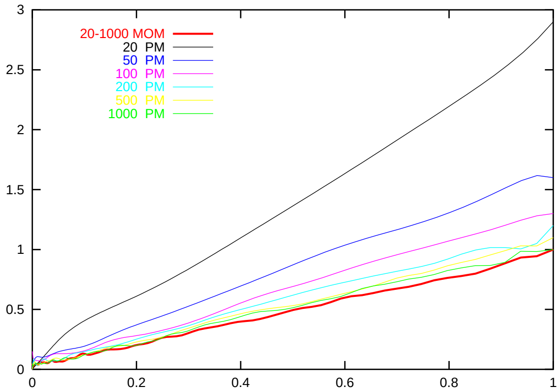
(b) Mean times (ms.)



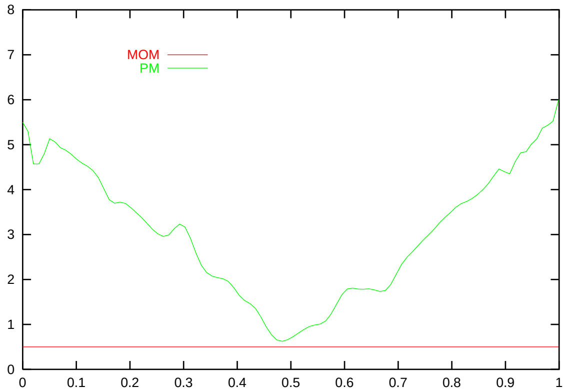
(c) Random test



(d) Linear test

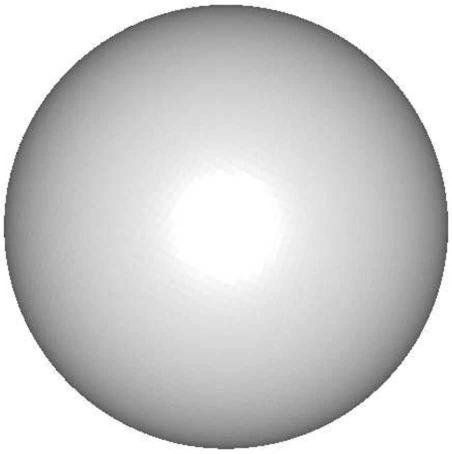


(e) Exponential test



(f) Retrieved LOD: 1403

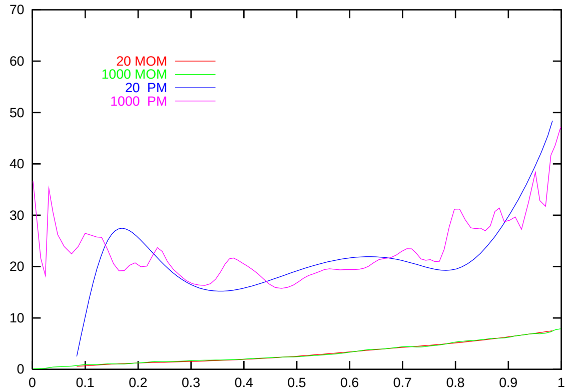
Figure 11: *Results obtained for the Cow model*



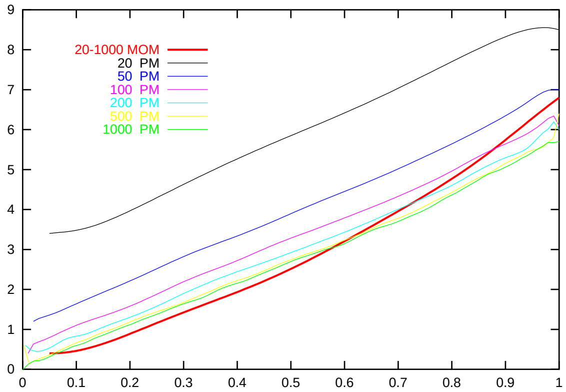
(a) Sphere

#LODs	Random Mom	Random Pm	Linear Mom	Linear Pm	Exponential Mom	Exponential Pm
20	2.9	21.2	3.0	6.0	1.7	5.1
50	3.4	21.0	2.8	4.0	1.5	2.9
100	3.2	22.9	2.8	3.3	1.5	2.2
200	3.1	22.5	2.8	3.0	1.5	1.8
500	3.0	22.3	2.8	2.8	1.5	1.6
1000	3.2	23.1	2.8	2.8	1.5	1.6

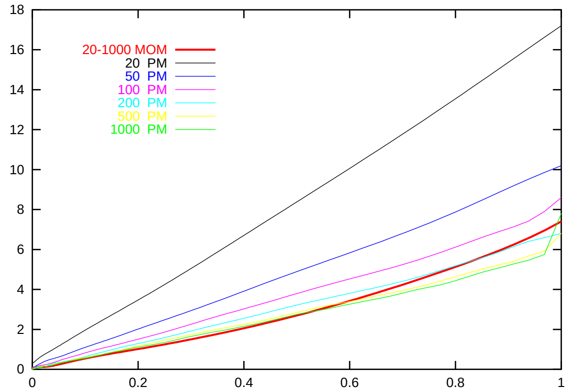
(b) Mean times (ms.)



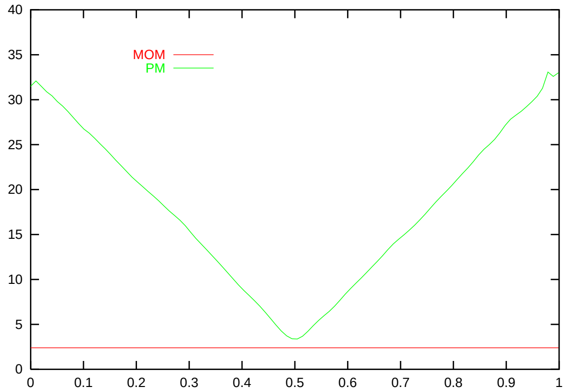
(c) Random test



(d) Linear test

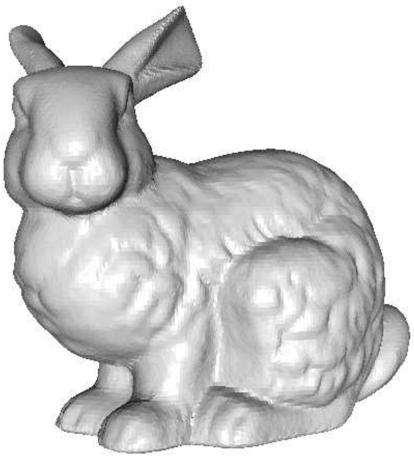


(e) Exponential test



(f) Retrieved LOD: 7632

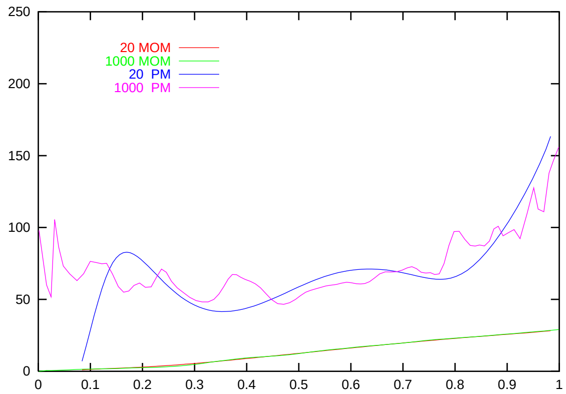
Figure 12: *Results obtained for the Sphere model*



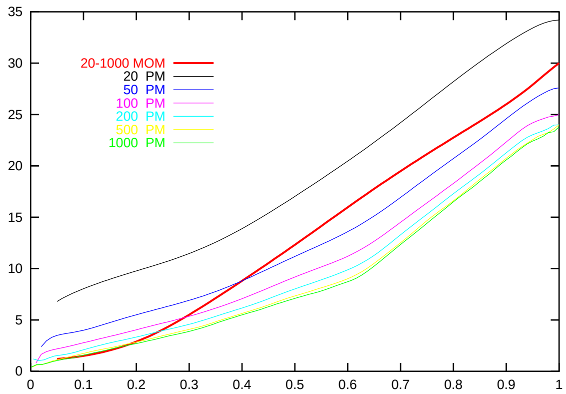
(a) Bunny

#LODs	Random		Linear		Exponential	
	Mom	Pm	Mom	Pm	Mom	Pm
20	12.5	66.1	13.7	18.9	7.0	15.8
50	14.5	67.2	13.2	12.9	6.8	8.7
100	13.4	71.4	13.1	10.9	6.7	6.6
200	13.1	69.5	13.0	9.9	6.6	5.6
500	12.6	68.5	13.0	9.3	6.6	5.0
1000	13.3	71.5	13.0	9.1	6.6	4.8

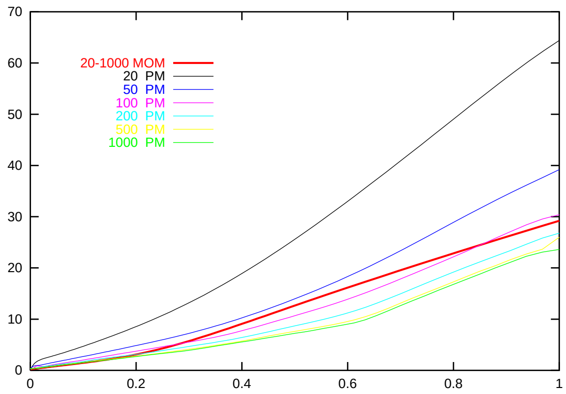
(b) Mean times (ms.)



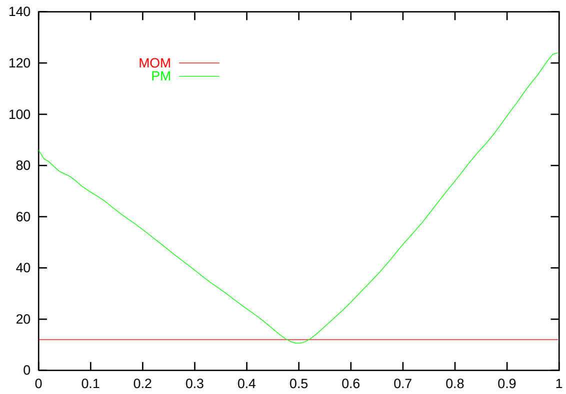
(c) Random test



(d) Linear test



(e) Exponential test



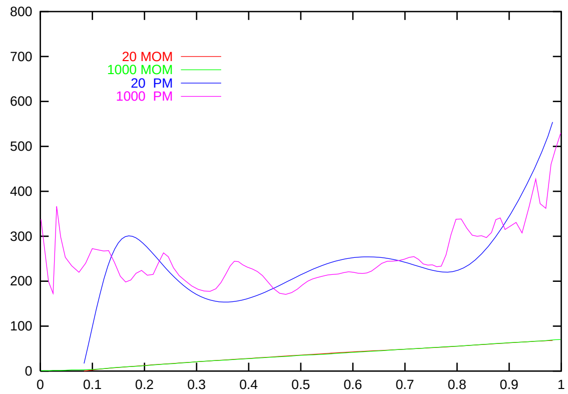
(f) Retrieved LOD: 16971

Figure 13: *Results obtained for the Bunny model*

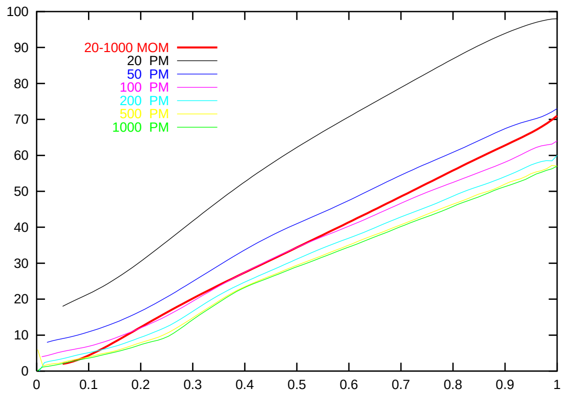
(a) Phone

#LODs	Random		Linear		Exponential	
	Mom	Pm	Mom	Pm	Mom	Pm
20	33.7	235.0	36.0	60.8	19.3	51.5
50	38.0	236.5	34.9	40.2	18.8	28.0
100	35.5	251.8	34.6	33.4	18.6	20.8
200	34.7	246.2	34.4	30.1	18.6	17.4
500	33.4	241.1	34.4	28.3	18.5	15.5
1000	35.0	251.2	34.3	27.8	18.5	14.8

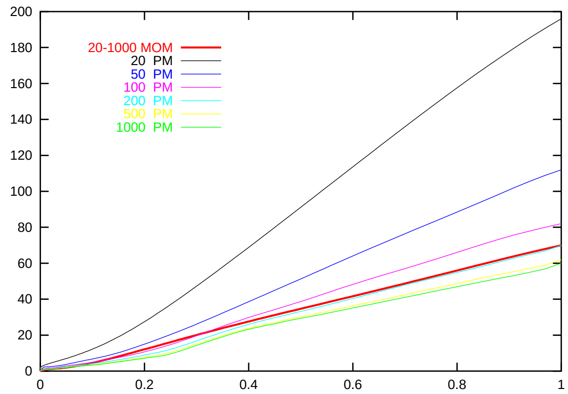
(b) Mean times (ms.)



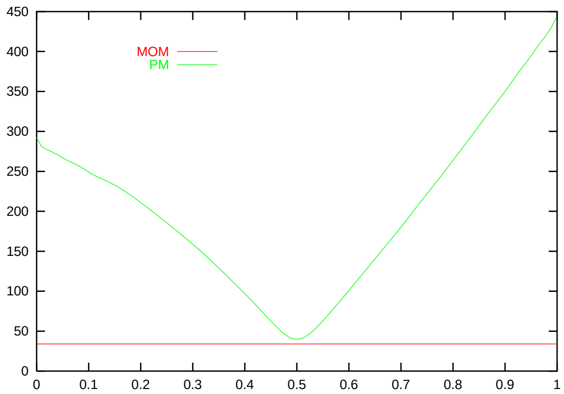
(c) Random test



(d) Linear test



(e) Exponential test



(f) Retrieved LOD: 41522

Figure 14: *Results obtained for the Phone model*