TOM: Totally Ordered Mesh

A multiresolution structure for time critical graphics applications

Eric Bouvier [*] Enrico Gobbetti[†]

CRS4
Center for Advanced Studies, Research and Development in Sardinia
Cagliari, Italy [‡]

Abstract

Tridimensional interactive applications are confronted to situations where very large databases have to be animated, transmitted and displayed in very short bounded times. As it is generally impossible to handle the complete graphics description while meeting timing constraint, techniques enabling the extraction and manipulation of a significant part of the geometric database have been the focus of many research works in the field of computer graphics. Multiresolution representations of 3D models provide access to 3D objects at arbitrary resolutions while minimizing appearance degradation. Several kinds of data structures have been recently proposed for dealing with polygonal or parametric representations, but where not generally optimized for time-critical applications. We describe the TOM (Totally Ordered Mesh), a multiresolution triangle mesh structure tailored to the support of time-critical adaptive rendering. The structure grants high speed access to the continuous levels of detail of a mesh and allows very fast traversal of the list of triangles at arbitrary resolution so that bottlenecks in the graphic pipeline are avoided. Moreover, and without specific compression, the memory footprint of the TOM is small (about 108% of the single resolution object in face-vertex form) so that large scenes can be effectively handled. The TOM structure also supports storage of per vertex (or per corner of triangle) attributes such as colors, normals, texture coordinates or dynamic properties. Implementation details are presented along with the results of tests for memory needs, approximation quality, timing and efficacy.

Keywords: multiresolution modeling, level-of-detail, adaptive rendering, time-critical graphics

1 Introduction

Synthetic scenes composed of thousands of highly detailed geometric models are rapidly becoming commonplace in computer graphics applications: from CAD systems, virtual reality, to interactive video games. Despite the continuous improvement in performance of CPUs and graphics accelerators, these scenes, exceeding a million of polygons, cannot be handled directly at interactive speeds even on high-end machines. An essential feature of scalable interactive 3D applications is thus the ability to trade rendering quality with speed, so as to meet the timing constraints associated to providing the illusion of continuous motion.

The traditional approach is to precompute a small number of independent level-of-detail (LOD) representations of each object composing the scene, choosing at run-time the best one based on a cost/benefit analysis [7, 25]. An improvement to this method consists in representing objects as multiresolution meshes, also known as continuous LODs. In this case, the LODs are stored in a dedicated data structure from which representations of the mesh can be extracted with any number of polygons.

In this paper, we discuss a multiresolution triangle mesh structure, the Totally Ordered Mesh structure (TOM), that has the following properties:

- The structure is compact. Without using specific compression techniques the memory overhead of the structure is about 8% of the full resolution mesh and becomes smaller if colors and texture coordinates are stored.
- The mesh can be accessed at any resolution in a short predictable time. The triangle traversal rates are in the order of a a few million triangles per second on common single processors machines (R10000 194 Mhz, Pentium II 300 MHz), which is comparable to what can be handled by high-end 3D accelerators.
- The data structure is designed to make optimal use of the immediate-mode vertex-face mesh rendering interfaces of modern graphics libraries. Rendering can be performed by first transferring to the graphics pipeline compact arrays of vertex attributes and then streaming through triangles to define the mesh connectivity with vertex indices.
- The quality and speed of the simplification methods that can be used to build the structure are competitive with other existing results.

These properties make this structure a good candidate for mesh representation in time-critical applications. The TOM structure has proven useful to support multiresolution scene rendering [9].

The rest of this paper presents related work and discusses our proposed structure. Implementation details are presented along with the results of tests for memory needs, approximation quality, timing and efficacy. We demonstrate the applicability of the structure to real-time applications with the implementation of a time-critical rendering engine that automatically selects the resolution of a set of objects represented with our multiresolution structure.

[*] Eric.Bouvier@crs4.it
[†] Enrico.Gobbetti@crs4.it
[‡] CRS4, VI Strada Ovest, Z.I. Macchiareddu, C.P. 94, I-09010 Uta (CA), Italy, http://www.crs4.it/vvr

2 Related work

2.1 Levels-of-detail

Many applications dealing with time-critical graphics are able to store a 3D model in a fixed number of independent resolutions (LODs) (e.g. OpenInventor [22] and VRML [29]). The main advantages of LODs are the simplicity of implementation and the fact that, as LODs are pre-calculated, the polygons can be organized in the most efficient way (triangle strips, display list), exploiting raw graphics processing speed with retained-mode graphics. The main drawbacks of this technique are related to its memory overhead, which severely limits in practice the number of LODs per object. As an example, representing an object at just the four LODs 100%, 75%, 50%, 25% would cause an overhead of 150% over the single resolution version. The small number of LODs limits the degrees of freedom of the display algorithm and might introduce visual artifacts due to the sudden changes of resolution between differing representations [13]. The use of accelerated retained mode can also be penalizing if the display lists cannot be entirely loaded in dedicated hardware memory [20, 21].

2.2 Multiresolution meshes

A multiresolution mesh is a compact data structure able to provide on demand a number of LODs that is proportional to the size of the mesh at full resolution. An extensive survey of multiresolution techniques is presented by Puppo et al. [24].

In the context of time-critical rendering applications, the desirable features a multiresolution mesh must possess are low memory overhead, to deal with large databases and maximize cache coherency, and fast triangle traversal rates at arbitrary resolutions, to provide fast adaptive rendering. While some of the existing algorithms are providing compact representations (e.g. progressive meshes [15]) or fast triangle access (e.g. hypertriangulations [4]), none of them provide a satisfactory compromise between space and time constraints, mainly because they were not designed for time-critical rendering applications.

The Progressive Mesh (PM) [15] representation is the most commonly used candidate structure. However, even if the representation is compact, it cannot be rendered directly, since it has first to be traversed to construct a single resolution mesh structure which can then be used for rendering [16]. Experimental results [16] are indicating a reconstruction rate of less than 200K triangles/sec on a Pentium Pro 200 Mhz. While this cost can be amortized on more than one frame if the single resolution meshes are cached, this is at the expense of memory. Moreover, exploiting per-object frame-to-frame coherency is only a partial solution for complex scenes, because of the discontinuity in scene complexity caused by objects entering into or exiting from the viewing frustum [7].

2.3 Dynamic simplification

An alternative to per-object LOD selection is to dynamically re-tessellate visible objects continuously as the user's viewing position shifts. As dynamic re-tessellation adds a run-time overhead, this approach is suitable when dealing with very large objects or static environments, when the time gained because of the better simplification is larger than the additional time spent in the selection algorithm. For this reason, this technique has been applied when the entire scene, or most of it, can be seen as a single multiresolution object from which to extract variable resolution representations.

The classic applications of dynamic re-tessellation techniques are in terrain visualization (see [14] for a survey). Xia et al. [31, 30] discuss the application of progressive meshes to scientific visualization. Luebke and Erikson [21] apply hierarchical dynamic simplification to large polygonal CAD models, by adaptively processing the entire database without first decomposing the environment into individual objects. To support interactive animated environments composed of many objects, we restrict ourselves to per-object constant resolution rendering.

3 Multiresolution triangle mesh structure

Because of their generality, triangle meshes are the most popular data structure for approximating surfaces in computer graphics applications. In applications where the main time-critical operation is surface rendering, the crucial optimizations are: the traversal of all the structure's triangles, and the retrieval of attributes at the vertices. On high-end systems it is particularly important to minimize the amount of data transferred from the application to the graphics pipe, since the host-to-pipe bandwidth limit can cause a performance bottleneck. A particularly effective approach is to construct primitives by indexing vertex data. Rendering can thus be performed by first transferring to the graphics pipeline compact arrays of vertex attributes and then streaming through triangles to define the mesh connectivity with vertex indices. Support for this type of direct-mode rendering has been recently introduced in high performance graphics libraries (e.g., OpenGL vertex array extension or Direct3D DrawIndexedPrimitive).

In this section we present a multiresolution triangle mesh structure that efficiently supports vertex packing and indexing. We named our data structure *Totally Ordered Mesh* (TOM), as it is based on totally ordered arrays of vertices and triangles. A similar structure has been independently developed by Guéziec et al. [12] for streaming geometry in VRML. Their structure, however, emphasizes the compaction of a small number of LODs in a single structure using a forest split approach, while we focus on continuous LODs for time-critical rendering applications.

3.1 Vertex substitution coding

Several algorithms have been recently published that simplify a polygonal mesh by iteratively contracting vertex pairs (e.g. [17, 10, 11, 15, 26, 8, 1]). A vertex pair contraction operation [8], denoted $(v_1, v_2) \rightarrow \tilde{v}$, replaces two vertices (v_1, v_2) with a single target point $\tilde{v}$ to which all the incident edges are linked, and removes any triangles that have degenerated into lines or points. The operation is quite general, and can express both edge-collapse and vertex clustering algorithms. The primary difference between vertex pair contraction algorithms lies in how the particular vertex pairs to be collapsed are chosen and in where the new vertices are positioned.

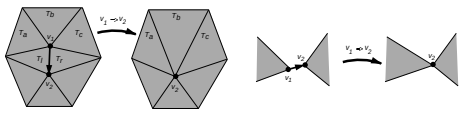

Figure 1: **Vertex substitution.** On the left, the substitution $v_1 \rightarrow v_2$ removes two triangles, as (v_1, v_2) is an edge. On the right, the substitution $v_1 \rightarrow v_2$ just connects two disjoint mesh regions. In both cases, all the simplification information can be retrieved from data stored at the vertex level in the original vertex list.

We define *vertex substitution*, denoted $v_1 \rightarrow v_2$, the restricted form of vertex pair contraction where the target point $\tilde{v}$ is constrained to be the second vertex of the pair v_2. Vertex substitution has the interesting property that it does not create any new vertex during the transformation. We can also consider that it does not require the creation of any new triangle: during the vertex substitution transformation, the surviving triangles can be interpreted as unchanged if we consider that the removed vertex is replaced at

lower resolutions by the remaining vertex of the contracted vertex pair (in figure 1, after the substitution $v_1 \rightarrow v_2$, vertex v_1 becomes equivalent to vertex v_2, triangles T_a, T_b and T_c are unchanged).

By iteratively applying vertex substitution, a triangle mesh can be reduced by removing one vertex and possibly some degenerated faces at a time (see figure 1). During the simplification process, no new vertex is inserted and the triangle list is not modified. Therefore, vertex substitution is an ideal transformation to create a multiresolution structure in a compact way, as all the simplification information can be encoded by just adding constant-size information to the vertex records in the original vertex list. The memory overhead introduced to store the multiresolution mesh is limited to the space required to store the vertex substitution history associated to vertex pair contraction. We encode a vertex substitution by associating to each vertex the reference to the vertex by which it is to be substituted. The resolution at which the substitution occurs need not be encoded as it is equal to the vertex index after sorting.

The necessary data structures are summarized as follows:

```
MESH = {
        ARRAY<VERTEX>   vertices
        ARRAY<TRIANGLE> triangles
}

TRIANGLE = {
        VERTEX_INDEX[3] vertices   -- triangle connectivity
}

VERTEX = {
        VERTEX_INDEX substitute
        ATTRIBUTES   attributes
}

ATTRIBUTES = {
        POINT3       position
        VECTOR3      normal
        ... color, texture, and other vertex attributes ...
}
```

The space overhead associated to the information added to the original single-resolution mesh is equal to $N_v * S_{vertex_index}$ where:

- N_v is the number of vertices;
- S_{index} is the size used to store the index to a vertex of the mesh

For a typical mesh of $N_t = 2N_v$ triangles, considering to use 32 bits to represent both a vertex index and a floating point number, the overhead associated to the above structure is less than 8% of the single full resolution mesh memory footprint when only position and normals are associated to vertices and becomes even smaller if other attributes such as colors and texture coordinates are present. We could still reduce the size of the mesh using mesh compression techniques [6], but this is not appropriate for time-critical rendering applications.

3.2 Total ordering for packing and traversal

As iterative vertex substitution does not modify vertex data and does not add new vertices, a total order can be defined both on the vertex list and on the triangle list based on the contraction resolution. Sorting according to this order after the simplification generates a compact and efficient data structure.

By ordering the vertex list, we obtain an optimally packed representation where the active vertices at resolution n are exactly the first n ones in the vertex array. In the case of graphics libraries supporting indexed mesh primitives, vertex data accessed at an arbitrary resolution may thus be communicated to the graphics pipeline for the rendering operation in the most efficient way, by passing a pointer to the first element in the contiguous block of memory storing the vertices together with the active vertex count.

Moreover, by ordering the triangle list, we have a way to iterate through the triangles that define the mesh at an arbitrary resolution in a time depending only on the number of active triangles and the lengths of the vertex substitution chains. Fortunately, these lengths are limited and relatively independent from the model size (see section 4). In any case the depth at full resolution is always 1 so that triangle traversal at full resolution is strictly linear. When resolution decreases, the traversal rate also decreases but slowly, because vertex substitution cannot, by definition, create too long chains for all the vertices. In fact, each vertex substitution $v_1 \rightarrow v_2$ increments by one the depth of all the vertex chains containing vertex v_1 but also keeps unchanged the length of all chains containing vertex v_2.

The following algorithm describes how to access all the triangles of a mesh for rendering at an arbitrary resolution:

```
--
-- 1. Send Vertex data
--

.. Send to pipeline the first ‘‘requested_resolution’’ vertices ..
in the vertex array

--
-- 2. Retrieve last vertex index at current resolution
--resolution is defined by the number of triangle ‘tri_count’
--
last_triangle := triangles[tri_count]
from
    v := last_triangle.vertices
until
    v[0]=v[1] or v[0]=v[2] or v[1]=v[2]
loop
    -- i receives the position of the max of the triple (v[0], v[1], v[2])
    i := PosMax(v[0], v[1], v[2])
    v[i] := v[i].substitute
    res_max := v[i]
end


--
-- 3. Send connectivity information
--
from
    current_triangle:= triangles.first
until
    current_triangle = last_triangle
loop
    v := current_triangle.vertices
    from
        i = 0
    until
        i > 2
    loop
        while
            v[i] > res_max
        do
            v[i] = v[i].substitute
        end
    end
    -- Send (v[0], v[1], v[2]) as connectivity information for
    -- the graphics pipeline

    current_triangle:= triangles.next
end
```

3.3 Attribute discontinuity

Associating rendering attributes to vertices is not always sufficient as it prevents modeling objects having discontinuities on their attributes. It is, for example, not possible to model sharp edges or to have adjacent triangles with distinct colors. A possible solution is to store the attributes by corners or using a wedge structure[16]. Both of the techniques have the drawback of introducing time and space overheads in the case of mostly continuous meshes: storing attributes at corners is space inefficient for vertices sharing the same attributes, and using a wedge structure introduces an indirection in accessing vertex data.

We prefer instead to manage discontinuities by duplicating vertices with different attributes and introducing degenerated triangles to preserve topology (see figure 2). As the number of discontinuous edges is generally very small compared to the total number of edges, the memory overhead is small, and no indirection is necessary to access vertex data.

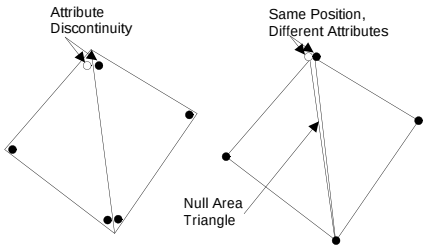

Figure 2: **Attribute discontinuity.** Discontinuities are handled introducing degenerate triangles.

3.4 Geomorphs

Smooth transitions between resolutions can be obtained by interpolating vertex data. As simplification proceeds by substituting only one vertex at each step, only the attributes for the last active vertex (i.e. the one that will be substituted at the next step) have to be interpolated. We currently implement this feature using linear interpolation for all data (see figure 3).

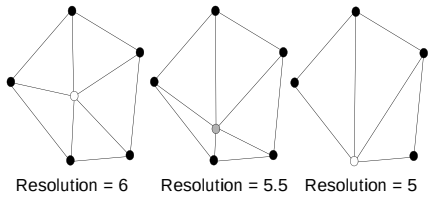

Figure 3: **Geomorph.** Smooth transitions between resolutions is obtained by interpolating vertex data along the last vertex substitution.

4 Implementation and results

A software library supporting the multiresolution triangle mesh structure described in this paper has been implemented and tested on Silicon Graphics and Windows NT machines. The results presented here were obtained on both architectures. The triangle meshes used in the evaluation were selected among those available in the public domain to enable comparison with other approaches. The characteristics of these meshes are summarized in table 1.

Mesh	Vertices	Triangles	Site
bunny	35942	69449	*www-graphics.stanford.edu*
fandisk	6871	12946	*www.research.microsoft.com/~hoppe*
cow	2915	5804	*www.cs.cmu.edu/~garland/cow.html*

Table 1: **Triangle meshes used in the tests**

4.1 Simplification

Our multiresolution structure requires that the mesh simplification method neither creates new triangles nor new vertices. As in [18], we have implemented mesh simplification schemes based on iterative vertex substitution in a generic framework of greedy algorithms for heuristic optimization. The generic greedy algorithm is parameterized by a *binary oracle*, deciding which vertex substitutions are legal, and a *fairness predicate*, which assigns priorities to all vertex substitutions in the legal candidate sets. The algorithm proceeds in a series of greedy steps until the candidate set is empty. In each greedy step, the best vertex substitution is removed from the candidate set, the mesh is modified accordingly, and new substitutions are reevaluated for the vertices affected by the mesh change. To speed-up these operations, the candidate set is implemented as a heap keyed on the priority, and references to the heap entries that have to be modified are associated to each of the active vertices. Building a multiresolution structure for a mesh with n faces has thus nearly $O(n)$ time complexity when using local binary oracles and fairness predicates, as time is dominated by priority computation and not by heap maintenance.

As in [1, 18] we have noticed in our various experiments that the most important factor to preserve quality during mesh simplification is the order in which operations are done, and that a good mesh reduction scheme can be achieved without inserting new vertices. Most of the methods that have been recently proposed can be adapted and restricted to vertex substitution and are implemented in this framework by just defining an appropriate binary oracle and fairness predicate. As noted by Garland and Heckbert [8], and confirmed by our experiments (see section 4), the effect of vertex position optimization in vertex contraction algorithms is only noticeable at very low resolutions. Since our context is multiresolution rendering, objects at low resolution do not need lots of details, as they contribute very little to the images. The loss in simplification quality is thus low, when compared to the performance gains provided by vertex substitution.

The possible vertex substitutions $v_1 \rightarrow v_2$ accepted by the binary oracle used for this paper, as in [8], are those for which (v_1, v_2) is an edge or $\|v_1 - v_2\| \leq t$, where t is a user-selected threshold parameter. This makes it possible to connect disjoint mesh regions during simplification. We obtained good results for the error (see section 4) by using a simple fairness predicate that computes a penalty coefficient with a simple memoryless technique similar to the one introduced by Lindstrom and Turk [19]. In all the examples we have used the following formula:

$$\epsilon_{v_1 \rightarrow v_2} = \Delta V_{v_1 \rightarrow v_2} + \|v_1 - v_2\| \Delta A_{v_1 \rightarrow v_2} + \delta Inv_{v_1 \rightarrow v_2}$$

where the penalty $\epsilon_{v_1 \rightarrow v_2}$ is a linear function of the volume variation $\Delta V_{v_1 \rightarrow v_2}$, the area variation $\Delta A_{v_1 \rightarrow v_2}$, the distance between vertices in a pair $\|v_1 - v_2\|$, and a penalty $\delta Inv_{v_1 \rightarrow v_2}$ which heavily penalizes substitutions that cause mesh inversion. We note that in our case the volume variation can be efficiently computed. As we apply vertex substitution, the volume variation is the sum of the tetrahedra based on the triangles that are modified during the contraction and that are passing by the vertex that is removed during the contraction.

When dealing with degenerated triangles to support attributes discontinuity, the way the penalty is computed must be adapted as simplifying a degenerated triangle doesn't change its volume, area and the distance between its vertices can be null. In this case, our solution is to introduce an additional coefficient measuring the distance between the vertices attributes. In the case of purely geometric meshes (as those in the results section), we simply add to the penalty a term $\Delta N_{v_1 \rightarrow v_2}$ proportional to the angle between the normals of the two vertices of the pair.

4.1.1 Quality

In order to assess the quality of the meshes representable by our multiresolution structure, we have compared the results obtained by

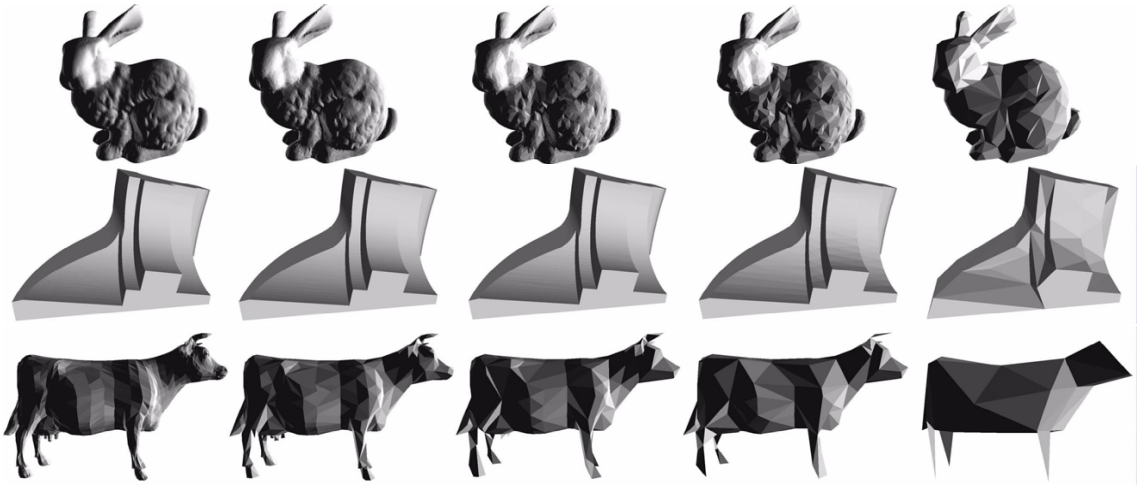

Figure 4: **Sequence of approximations of the *bunny*, *cow*, and *fandisk* datasets.** The original models on the left have full resolution. The approximations to the right have 20%, 10%, 5%, and 1% triangles respectively.

our simplification algorithm with those generated with other state-of-the art techniques. The algorithms considered are Mesh Decimation [27], Progressive Meshes [15], Simplification Envelopes [5], Multiresolution Decimation [2], and Quadric Error Metric [8]. The comparison has been done using the publicly available *Metro* tool [23], which measures surface deviation and is thus biased towards simplification algorithms which use that criterion to guide simplification. Figure 5 and table 2 summarize the results obtained on the *bunny* model. The source for data on competing algorithms is reference [3]. Despite the constraints imposed by the multiresolution structure and the simple memoryless binary oracle and fairness predicate, our simplification algorithm obtains results comparable to those of recently published simplification methods. The visual quality of the approximations is illustrated in figure 4.

Vertices	Triangles	Maximum error	Average error
17418 (50%)	34653	0.47433	0.01009
8079 (25%)	16006	0.83984	0.01631
3484 (10%)	6873	0.79855	0.02583
1742 (5%)	3428	1.04764	0.04387
697 (2%)	1361	1.02499	0.09544
349 (1%)	675	1.78891	0.18555
175 (0.5%)	337	2.41363	0.34466

Table 2: **Geometric error measurements for the *bunny* model at various resolutions.** Errors are % of the dataset bounding box diagonal.

4.1.2 Time

The performances that we have obtained when building our multiresolution structure from the full resolution mesh are presented in table 3. The simplification time is linear with the full resolution triangle count, and exceeds the rate of 1000 triangles/second on all models that we have tested. The simplification time is thus comparable to that of other approaches (see [3, 19]), even though, in our case, the output is a full multiresolution representation instead of a single simplified model. This shows that little overhead is associated to constructing the structure.

Mesh	Time (seconds)	Triangles/second
bunny	50.6	1370.5
fandisk	10.1	1286.8
cow	3.9	1482.2

Table 3: **Multiresolution construction time.**

4.2 Memory

Table 4 summarizes the storage requirements of the TOM structure for the test meshes, compared to the original mesh in face-vertex form, to a typical LOD representation with the six levels of details 100%, 50%, 25%, 12%, 6%, 3%, and to a progressive mesh (PM) representation using the memory-resident data structure presented in [16], simplified by removing face-level attributes to make comparison fair. As the PM representation cannot be rendered directly, but has first to be traversed to construct a single resolution mesh structure [16], we provide for this representation the minimum and maximum memory required, corresponding to rendering the mesh at the lowest, respectively highest, possible resolution. All size estimations assume that the mesh contains only normals as vertex attributes, and that 32 bits are used for both integer and floating point data.

Mesh	*bunny*	*fandisk*	*cow*
Face-Vertex	1696 (100%)	320 (100%)	140 (100%)
TOM	1840 (108%)	348 (108%)	151 (108%)
LOD	3521 (213%)	666 (213%)	290 (212%)
PM-0%	1966 (119%)	376 (120%)	159 (117%)
PM-100%	4436 (268%)	840 (269%)	364 (267%)

Table 4: **Minimum storage needs for a rendering application.** Sizes in Kb, percentages are with respect to the mesh in face-vertex form.

As we can see from table 4, the TOM multiresolution structure is the most compact, with an overhead of only 8% with respect to the single resolution mesh in face-vertex form. The overhead fraction is further reduced when associating more attributes at each vertex (e.g. color, texture coordinates).

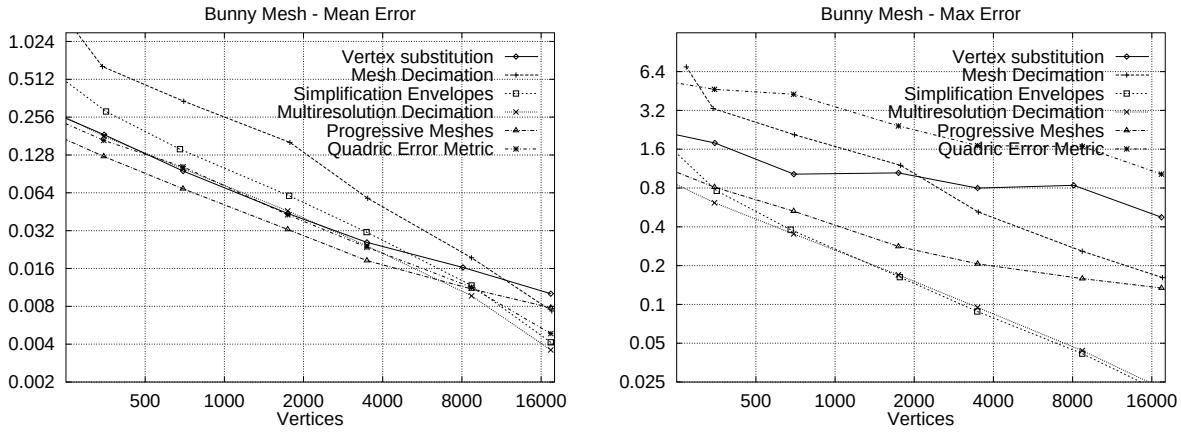

Figure 5: **Mean and maximum geometric error.** Errors are % of the dataset bounding box diagonal. The source for data on competing algorithms is reference [3]

.

4.3 Traversal and rendering

As we are focusing on time-critical scene rendering applications, the most important results are relative to the triangle traversal rate through our multiresolution structure. The overhead with respect to traversing a single resolution mesh is only dependent on the lengths of the vertex substitution chains. Figure 6 presents the overhead associated to traversing the *bunny*, *fandisk*, and *cow* meshes at various resolutions. As we can see, the results on different meshes are very similar. The overhead grows slowly with the simplification ratio, and in all cases, never exceeds 75%. It is thus possible to traverse all active triangles while retrieving vertex attributes at a speed sufficient to feed the 3D graphics pipeline even on high-end 3D accelerators.

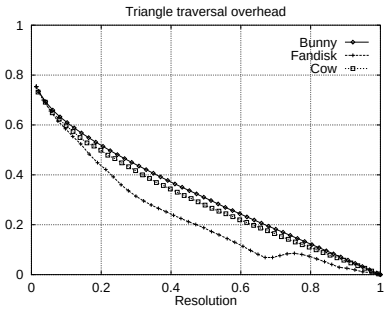

Figure 6: **Multiresolution traversal overhead.** Ratio of extra vertices traversed to render meshes at multiple resolutions. Resolution expressed as fraction of vertices.

Figure 7 presents the data structure traversal and rendering performance obtained with the multiresolution structure for the *cow* mesh. Similar results were obtained for the other meshes and are not presented here. For the benchmarks we used two directional lights, Gouraud shading, and a viewing configuration compressing the mesh to a 50x50 pixel area, to avoid fill-rate bottlenecks. Our current implementation renders meshes with the strict OpenGL core command set as a sequence of independent triangles whose vertices are specified using `glNormal/glVertex` calls. Rendering with vertex index extension is slower, as rendering indexed primitives is only emulated in the drivers used. We expect a performance improvement with native implementation the `EXT_vertex_array` extension.

As we can see, even using standard OpenGL, the rendering performance at all resolutions for the multiresolution version is similar to the limit given by rendering a streamlined version of the mesh. In both cases, the meshes are rendered at a constant speed of about 800 KTris/second at all resolutions, showing that our linear cost heuristics is a good approximation of the rendering behavior. Triangle traversal rates on the multiresolution structure, measured by having the traversal routine call empty vertex/normal procedures are well above the rates obtained with the rendering code for all resolutions, since they range from 3.7 MTris/second to over 10 MTris/second.

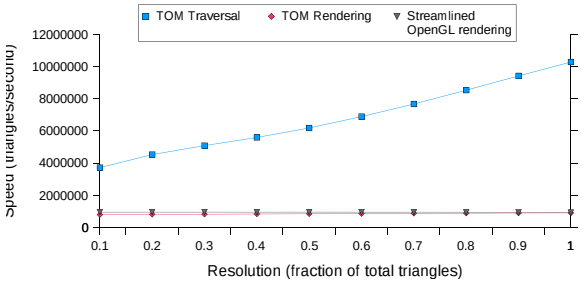

Figure 7: **Multiresolution traversal and rendering performance.** Timing statistics for the *cow* mesh. Experimental measures on a SGI 320, single 500 MHz Pentium III with 512 Kb L2 cache, 256 Mb RAM, Cobalt graphics. Rendering environment: two directional lights, one material per object, Gouraud shading, independent triangles, one normal per vertex

4.4 Time-critical rendering application

We have developed an application that demonstrates the performances of our structure in a time-critical context. The rendering engine is based on a feedback algorithm that automatically selects the resolution of the objects of the scene. The technique used is based on a simple *stress management* approach [7, 25]. Two state variables, the system $load$, defined as the fraction of expected time actually used to render the frame, and the system $stress$, a function of load over time, are used to increase or reduce the number of rendered triangles so as to meet the timing deadlines. At system start, the $stress$ is initialized at one. At the beginning of each frame, the

system selects the resolution at which to render the visible objects by linearly mapping to the available resolution range their bounding box screen coverage scaled by $1/stress$. Thus, when the stress is high, coarser models are chosen so as to reduce future system load. At the end of a frame N, $load_N$ and $stress_N$ are updated as follows:

$$load_N = {T_N}^{actual}/T^{expected}$$

$$stress_N = \begin{cases} stress_{N-1} * (1-\alpha) & \text{if } load_N < load^{min} \\ stress_{N-1} * (1+\alpha) & \text{if } load_N > load^{max} \\ stress_{N-1} & \text{otherwise} \end{cases}$$

where ${T_N}^{actual}$ is the measured time for frame N, $T^{expected}$ is the user-specified target drawing time, $\alpha < 1$ is the stress update factor, and $]load^{min}, load^{max}[$ defines an hysteresis band that reduces unwanted oscillations.

The graphs of figure 9 illustrate the approach with data gathered during a fly-through of the test scene of figure 8 with a target frame rate of 10 frames/second. The results presented here were obtained on a Silicon Graphics Onyx with 2 MIPS R10000 194 MHz CPU, primary data cache size of 32 Kbytes, secondary unified instruction/data cache size of 1 Mbyte, main memory size of 1Gb, and an InfiniteReality graphics board with two raster managers with 64 Mb of texture memory.

Despite the enormous and abrupt changes in potential complexity due to objects entering into and exiting from the viewing frustum, we see that the load is always maintained below one, and thus that the system was able to meet the timing constraints thanks to the fast random access times of our structure.

The good results obtained despite the simple feedback adaptation technique are due to the continuous nature of the multiresolution models, which overcomes the problems experienced when using discrete LODs [7, 25]. The wide range of continuous variation of resolution for objects in a scene is illustrated by the close-up view of figure 10.

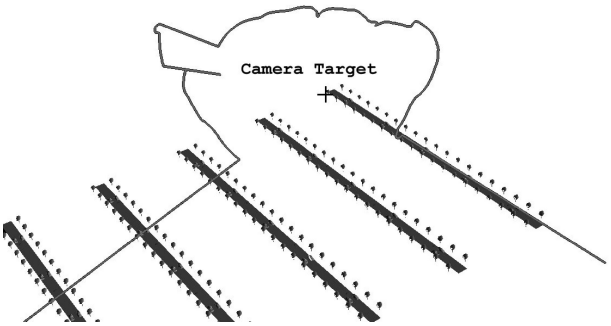

Figure 8: **Adaptive rendering test scene**. The total scene complexity is 1.15 Mtriangles. The fly-through path analyzed is traversed from left to right.

5 Conclusions

We have presented a simple multiresolution data structure for representing continuous levels-of-detail of triangle meshes. The structure is compact, requiring only a small overhead over the single full resolution mesh, provides fast triangle and vertex traversal rates at arbitrary resolution, and can make optimal use of immediate-mode vertex-face mesh rendering interfaces.

Any simplification method reducible to a sequence of vertex substitutions can be used to build the structure. The quality and speed of the simple memoryless simplification technique we have illustrated are competitive with other existing results.

These characteristics makes it very appropriate for mesh representation in time-critical scene rendering applications. We have integrated the structure in a collaborative CAD system [28] and in an experimental time-critical rendering engine [9]. Our future work will focus on extending the structure to support view-dependent resolution optimization.

Acknowledgments

The authors would like to thank Hugues Hoppe, Michael Garland, and the Stanford graphics group for making benchmark datasets available, and the Visual Computing Group at I.E.I. for the development of the *Metro* tool. We also greatly appreciate the help received from Marco Agus and Fabio Bettio in taking experimental measures.

This research is partially sponsored by the European project CAVALCADE (ESPRIT contract 26285). We also acknowledge the contribution of Sardinian regional authorities.

References

[1] CAMPAGNA, S., KOBBELT, L., AND SEIDEL, H.-P. Efficient decimation of complex triangle meshes. Technical Report 3, Computer Graphics Group, University of Erlangen, Germany, 1998.

[2] CIAMPALINI, A., CIGNONI, P., MONTANI, C., AND SCOPIGNO, R. Multiresolution decimation based on global error. *The Visual Computer 13*, 5 (1997), 228–246. ISSN 0178-2789.

[3] CIGNONI, P., MONTANI, C., AND SCOPIGNO, R. A comparison of mesh simplification algorithms. *Computers and Graphics 22*, 1 (Jan. 1998), 37–54.

[4] CIGNONI, P., PUPPO, E., AND SCOPIGNO, R. Representation and visualization of terrain surfaces at variable resolution. *The Visual Computer 13*, 5 (1997), 199–217. ISSN 0178-2789.

[5] COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., BROOKS, F., AND WRIGHT, W. Simplification envelopes. In *SIGGRAPH '96 Proc.* (Aug. 1996), pp. 119–128. URL: http://www.cs.unc.edu/~geom/envelope.html.

[6] DEERING, M. Geometry compression. In *SIGGRAPH '95 Proc.* (Aug. 1995), ACM, pp. 13–20.

[7] FUNKHOUSER, T. A., AND SÉQUIN, C. H. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (SIGGRAPH '93 Proc.)* (1993).

[8] GARLAND, M., AND HECKBERT, P. S. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Proc.* (Aug. 1997), pp. 209–216. URL: http://www.cs.cmu.edu/~garland/quadrics.

[9] GOBBETTI, E., AND BOUVIER, E. Time-critical multiresolution scene rendering. In *IEEE Visualization '99* (Oct. 1999), IEEE.

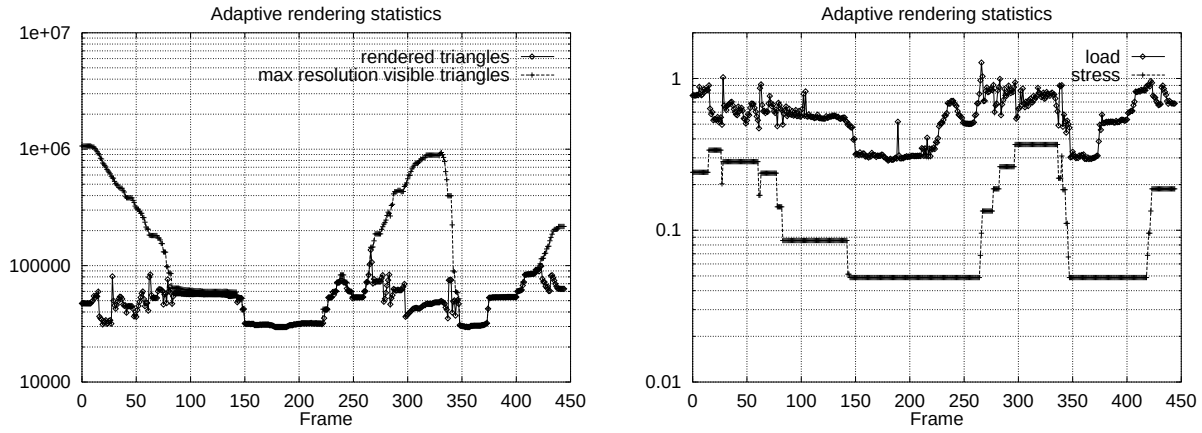

Figure 9: **Adaptive rendering statistics**. Data gathered during fly-through illustrated in figure 8 with $T^{expected} = 100ms$, $\alpha = 0.4$, $load^{min} = 0.5$ and $load^{max} = 1.0$. Notice the large increase/decrease in scene complexity due to culling. Results obtained on a Silicon Graphics Onyx with 2 MIPS R10000 194 MHz CPU, primary data cache size of 32 Kbytes, secondary unified instruction/data cache size of 1 Mbyte, main memory size of 1Gb, and an InfiniteReality graphics board with two raster managers with 64 Mb of texture memory.

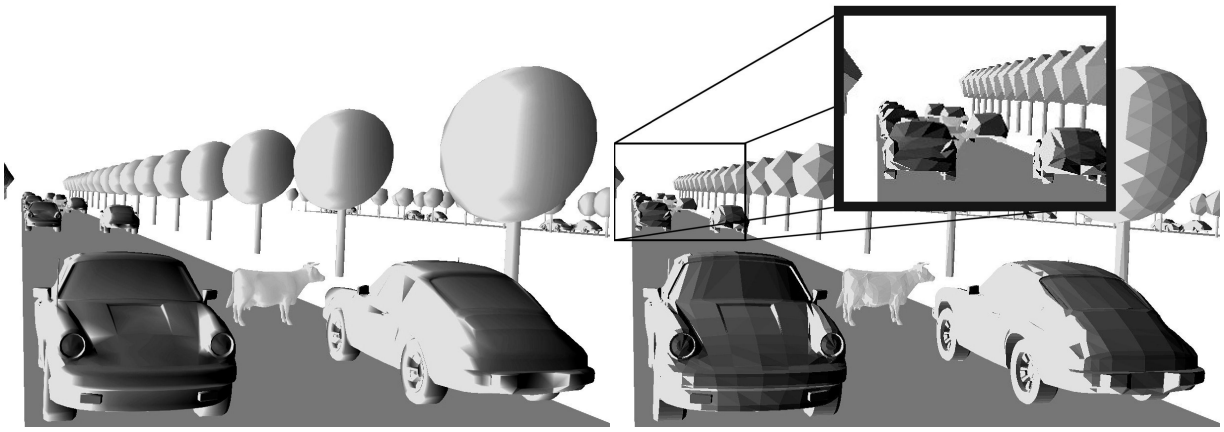

Figure 10: **Adaptive rendering test scene**. Close-up view during fly-through. The image to the left is the one presented to the viewer, while the image to the right is rendered with flat shading to show the variation in levels of detail, not noticeable otherwise. Results obtained on a Silicon Graphics Onyx with 2 MIPS R10000 194 MHz CPU, primary data cache size of 32 Kbytes, secondary unified instruction/data cache size of 1 Mbyte, main memory size of 1Gb, and an InfiniteReality graphics board with two raster managers with 64 Mb of texture memory.

[10] GUÉZIEC, A. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)* (November 1995), pp. 132–139.

[11] GUÉZIEC, A. Surface simplification inside a tolerance volume. Tech. rep., Yorktown Heights, NY 10598, Mar. 1996. IBM Research Report RC 20440, URL: http://www.watson.ibm.com:8080/search_paper.shtml.

[12] GUÉZIEC, A., TAUBIN, G., HORN, B., AND LAZARUS, F. A framework for streaming geometry in VRML. *IEEE Computer Graphics and Applications 19*, 2 (Mar./Apr. 1999), 68–78.

[13] HECKBERT, P. S., AND GARLAND, M. Multiresolution modeling for fast rendering. In *Proc. Graphics Interface '94* (Banff, Canada, May 1994), Canadian Inf. Proc. Soc., pp. 43–50. URL: http://www.cs.cmu.edu/~ph.

[14] HECKBERT, P. S., AND GARLAND, M. Survey of polygonal surface simplification algorithms. Tech. rep., CS Dept., Carnegie Mellon U., to appear. URL: http://www.cs.cmu.edu/~ph.

[15] HOPPE, H. Progressive meshes. In *SIGGRAPH '96 Proc.* (Aug. 1996), pp. 99–108. URL: http://www.research.microsoft.com/research/graphics/hoppe/papers.html.

[16] HOPPE, H. Efficient implementation of progressive meshes. *Computers and Graphics 22*, 1 (January 1998), 27–36.

[17] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Mesh optimization. In *SIGGRAPH '93 Proc.* (Aug. 1993), pp. 19–26. URL: http://www.research.microsoft.com/research/graphics/hoppe/papers.html.

[18] KOBBELT, L., CAMPAGNA, S., AND SEIDEL, H.-P. A general framework for mesh decimation. In *Proceedings of the 24th Conference on Graphics Interface (GI-98)* (San Francisco, June18–20 1998), W. Davis, K. Booth, and A. Fourier, Eds., Morgan Kaufmann Publishers, pp. 43–50.

[19] LINDSTROM, P., AND TURK, G. Fast and memory efficient polygonal simplification. In *Proceedings IEEE Visualization'98* (1998), IEEE, pp. 279–286.

[20] LUEBKE, D. Hierarchical structures for dynamic polygonal simplification. TR 96-006, Department of Computer Science, University of North Carolina at Chapel Hill, 1996.

[21] LUEBKE, D., AND ERIKSON, C. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), T. Whitted, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 199–208. ISBN 0-89791-896-7.

[22] OPEN INVENTOR ARCHITECTURE GROUP. *Open Inventor C++ Reference Manual: The Official Reference Document for Open Systems*. Addison-Wesley, Reading, MA, USA, 1994.

[23] P. CIGNONI, C. R., AND SCOPIGNO, R. Metro: measuring error on simplified surfaces. Tech. rep., Istituto I.E.I.-C.N.R., Pisa, Italy, Jan. 1996. Technical Report B4-01-01-96, URL: `http://miles.cnuce.cnr.it/cg/metro.img.html`.

[24] PUPPO, E., AND SCOPIGNO, R. Simplification, LOD, and multiresolution: Principles and practice. Eurographics tutorial notes, 1997.

[25] ROHLF, J., AND HELMAN, J. IRIS performer: A high performance multiprocessing toolkit for real–Time 3D graphics. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)* (July 1994), A. Glassner, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, ACM Press, pp. 381–395. ISBN 0-89791-667-0.

[26] RONFARD, R., AND ROSSIGNAC, J. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum 15*, 3 (Aug. 1996). Proc. Eurographics '96.

[27] SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proc.) 26*, 2 (July 1992), 65–70.

[28] TORGUET, P., BALET, O., GOBBETTI, E., JESSEL, J.-P., DUCHON, J., AND BOUVIER, E. Cavalcade: A system for collaborative prototyping. In *Proc. International Scientific Workshop on Virtual Reality and Prototyping* (May 1999). Conference held in Laval, France, June 3-4.

[29] VRML 97, International Specification ISO/IEC IS 14772-1, Dec. 1997.

[30] XIA, J. C., EL-SANA, J., AND VARSHNEY, A. Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (Apr. 1997), 171–183.

[31] XIA, J. C., AND VARSHNEY, A. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96* (Oct. 1996), IEEE. ISBN 0-89791-864-9.