Time-critical Multiresolution Scene Rendering

Enrico Gobbetti* Eric Bouvier †

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

Abstract

We describe a framework for time-critical rendering of graphics scenes composed of a large number of objects having complex geometric descriptions. Our technique relies upon a scene description in which objects are represented as multiresolution meshes. We perform a constrained optimization at each frame to choose the resolution of each potentially visible object that generates the best quality image while meeting timing constraints. The technique provides smooth level-of-detail control and aims at guaranteeing a uniform, bounded frame rate even for widely changing viewing conditions. The optimization algorithm is independent from the particular data structure used to represent multiresolution meshes. The only requirements are the ability to represent a mesh with an arbitrary number of triangles and to traverse a mesh structure at an arbitrary resolution in a short predictable time. A data structure satisfying these criteria is described and experimental results are discussed.

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

1 Introduction

The steadily increasing complexity of graphics applications presents important challenges to application developers. This is particularly true for highly interactive 3D programs, such as visual simulations and virtual environments, with their inherent focus on interactivity, low-latency, and real-time processing. Many application domains of interactive 3D graphics are characterized by the need to visualize in real-time dynamic graphics scenes with a complex geometric description. These scenes, exceeding millions of polygons and hundreds of objects, cannot be handled directly at interactive speeds even on high-end machines. As there is no upper bound on the complexity of a scene visible from a specific viewpoint, meeting the performance requirements dictated by the human perceptual system requires the ability to trade rendering quality with speed. Ideally, this time/quality conflict should be handled with adaptive techniques, to cope with the wide range of viewing conditions while avoiding worst-case assumptions.

Recently, a number of efficient adaptive algorithms have been introduced that incrementally adapt the rendering complexity of large-scale surfaces (e.g terrains [2, 13] or large organic surfaces [10]). Many types of graphics scenes, however, often contain a large number of distinct and possibly animated small-scale objects (e.g. rigid body simulations, virtual engineering prototypes [22]). The traditional approach to render these scenes in a time-critical setting is to pre-compute a small number of independent level-of-detail (LOD) representations of each object composing the scene, and to switch at run-time between the LODs.

In this paper, we propose to model 3D scenes as collections of multiresolution meshes and to choose the resolution of each mesh by solving a continuous constrained optimization problem (i.e. the maximization of scene quality under timing constraints). This time-critical multiresolution scene rendering framework improves over current solutions in the following areas:

- **Ability to meet timing constraints.** In contrast to current static or feedback algorithms, our technique aims at guaranteeing a uniform, bounded frame rate even for widely changing viewing conditions; the technique is thus appropriate in a time-critical setting and enables the use of prediction to reduce perceived lag [24];

- **Scene and hardware independence.** Both the desired quality and the hardware behavior are modeled by customizable heuristics. This makes it possible to incorporate context-sensitive quality constraints and makes the algorithm independent from the specific hardware on which the application is running, which is of primary importance for distributed multi-platform graphics applications visualizing a shared model.

- **Measurable image quality and distance from optimum.** Since objects are modeled as multiresolution meshes, nearly imperceptible transitions between resolutions are obtained at no cost. Furthermore, our algorithm provides for each image both a quality measure and a bound on the distance in quality from the optimal one. The typical accuracy of our solutions is less than 5%, which is an order of magnitude better than what can be guaranteed by current combinatorial optimization approaches [3, 17];

- **Data structure independence.** The optimization algorithm is independent from the particular data structure used to represent multiresolution meshes. The only requirements are the ability to represent a mesh with an arbitrary number of triangles and to traverse the structure at an arbitrary resolution in a short predictable time. A data structure satisfying these criteria is presented in section 6.

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

2 Background and related work

2.1 Levels-of-detail

Many applications dealing with time-critical graphics include the possibility to store a 3D model in a fixed number of independent resolutions (e.g. OpenInventor [18] and VRML [23]). 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. This constraint might introduce visual artifacts due to the sudden changes of resolution between differing representations [8] and, more importantly, limits the degrees of freedom of the LOD selection algorithm.

Run-time LOD selection is typically done using static heuristics or feedback algorithms. Static heuristics (e.g. distance-based [23] or coverage-based [18] LOD mappings) are not adaptive and are therefore inherently unable to produce uniform frame rates, while feedback algorithms, which adaptively vary LOD mappings based on past rendering times (e.g. [20]) suffer of unavoidable overshoot and oscillation problems during visualization of discontinuous virtual environments.

As demonstrated by Funkhouser and Séquin [3], to guarantee bounded frame times, predictive algorithms that optimize LOD selection based on an estimate of the time to render the frame must be used. Having a *guarantee* on the total maximum lag of the application is a necessary precondition for using prediction techniques for lag reduction [24] Unfortunately, the combinatorial optimization problem associated to LOD selection is equivalent to a version of the Multiple Choice Knapsack Problem [3, 17], which is NP-complete, and approximation algorithms that cannot guarantee optimality have to be used. Current state-of-the-art techniques (Funkhouser and Séquin's greedy algorithm [3] and Mason and Blake's [17] incremental technique) produce a result which is only guaranteed to be half as good as the optimal solution.

2.2 Dynamic simplification

An alternative to per-object LOD selection is to dynamically re-tessellate visible objects continuously as the 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 [9] for a survey). Hoppe [11] introduced view-dependent simplification of progressive meshes, applying it to the visualization of single large objects. Xia et al. [26, 25] discuss the application of progressive meshes to scientific visualization. Luebke and Erikson [16] 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 Overview of the approach

Our approach relies upon a scene description in which objects are represented as multiresolution triangle meshes, i.e. compact data structures able to efficiently provide on demand a triangle mesh approximation of an object with the requested number of faces. At each frame, we aim to find within a fixed short time the triangle mesh representation for each potentially visible object that produces the "best quality" image within the target frame time. This is done in an optimization phase which takes as input the set of potentially visible objects determined by a culling algorithm (e.g. bounding box or occlusion culling) and selects as output the list of triangle meshes to be rendered.

More formally, a triangle mesh is a piecewise linear approximation of a 3D shape that can be denoted by a tuple $M = (K, V, A)$, where K is a simplicial complex specifying the connectivity of the mesh simplices (the adjacency of the vertices, edges, and faces), $V = \{v_1, v_2, \ldots, v_m\}$ is the set of vertex positions defining the shape of the mesh in $\Re^3$, and $A = \{a_1, a_2, \ldots, a_m\}$ is the set of attributes associated to the vertices of M. Both the quality of approximation and the rendering complexity are dependent upon the sizes of K and V. A multiresolution representation mr_M for a mesh M with n_v vertices and n_t triangles can be seen as a function that takes as input the desired resolution $r \in [0, 1] \subset \Re$ and produces as output another mesh $M^r = mr_M(r)$ that approximates the same shape with $n_t^{(\text{desired})} = \lfloor r n_t \rfloor$ faces.

At each frame, the culling algorithm produces thus a a parameterized representation $S(\mathbf{r})$ of the visible objects set, which associates to each parameter value $\mathbf{r} \in [0, 1]^n$ an actual set of triangle meshes $S(\mathbf{r}) = \{mr_{M_1}(r_1), mr_{M_2}(r_2), \ldots, mr_{M_n}(r_n)\}$. Our goal is to find the optimal parameter vector $\mathbf{r}^\star$ for a viewing configuration W. To this end, we define two heuristics for the visible object set: a $cost(W, S(\mathbf{r}))$ heuristic, which estimates the time required to render a scene containing the visible objects at resolution $\mathbf{r}$, and a $benefit(W, S(\mathbf{r}))$ heuristic, which estimates the quality of the rendered image.

Even though the multiresolution representation is discontinuous, as there are only n_t possible values for a base mesh with n_t triangles, we can safely assume that benefit and cost heuristics are smooth. This simplifying assumption, at the core of our approach, introduces an error which is clearly negligible for sufficiently large values of n_t with respect to the error intrinsically induced by the use of heuristics. Furthermore, it should also be noticed that, even though this fact is not used in our implementation, smooth transitions between LODs, and thus smooth benefit heuristics, can be obtained using geomorphs.

Using this formalism, our approach to predictive adaptive rendering is stated as follows:

$$\begin{array}{ll} \text{Maximize:} & benefit(W, S(\mathbf{r})) \\ \text{Subject to:} & cost(W, S(\mathbf{r})) \leq t^{(\text{desired})} \\ & \mathbf{r} \succeq \mathbf{0} \\ & \mathbf{r} \preceq \mathbf{1} \end{array} \tag{1}$$

where $\succeq$ and $\preceq$ denote componentwise inequality, $\mathbf{0}$ and $\mathbf{1}$ are constant vectors and $t^{(\text{desired})}$ is the target display time. The cost and benefit heuristics, the optimization algorithm and the multiresolution mesh representation are discussed in the following sections.

4 Cost and benefit heuristics

4.1 Cost heuristics

The $cost(W, S(\mathbf{r}))$ heuristic provides an estimation of the time necessary to render in a viewing configuration W (camera, lights), a scene composed of the multiresolution objects present in S at resolutions $\mathbf{r}$. Time-critical renderers are typically running on top of a pipelined graphics hardware implementing a Z-buffer algorithm.

Scene display starts with an initialization phase (initial setup, buffer clears), followed by the sequential rendering of each of the meshes, and finishes by a finalization phase (buffer swap). Initialization and finalization time can be considered constants, while, assuming a fast enough CPU and an efficient multiresolution mesh representation, mesh rendering is dominated either by the time to define rendering attributes and process all the triangles, or by the time to fill the covered pixels, depending on where the pipeline bottleneck is. On most hardware configurations, the time to define rendering attributes and process all the triangles is just dictated by the speed of the graphics pipe-line for all operations before rasterization. On very high-end graphics systems, actually fetching triangles from the multiresolution structure may become the dominant phase. In both cases, however, we assume that the cost remains linear in the number of triangles and we thus only need to determine the "speed" of the dominant phase for the prediction of the rendering time. In other words, the cost of rendering a multiresolution mesh M at resolution r can be estimated as follows:

$$T^{(\text{setup})} + max \left\{ \begin{array}{l} T^{(\text{tri})} r \cdot maxtri(M) \\ T^{(\text{pix})} coverage(M, W)) \end{array} \right\} \quad (2)$$

where $maxtri(M)$ is the maximum number of triangles for mesh M, $T^{(\text{setup})}$ is the time associated to setup the rendering environment for an object (e.g. material binding time for OpenGL), $T^{(\text{tri})}$ is the time to send a triangle through the pipeline (i.e. the maximum between the time to fetch a triangle from the multiresolution structure and that of processing it without rasterization), $T^{(\text{pix})}$ is the time to fill a pixel, and $coverage(M, W)$ is an estimation of the number of pixels covered by mesh M when rendered with a viewing configuration W. From equation 2, we can derive the minimal resolution $r^{(\text{min})}$ under which a reduction in resolution (and therefore possibly in quality) does not reduce rendering time:

$$r^{(\text{min})} = \frac{T^{(\text{pix})} coverage(M, W)}{T^{(\text{tri})} maxtri(M)} \quad (3)$$

The cost heuristics can thus be modeled as:

$$\begin{array}{l} cost(W, S(\mathbf{r})) = T^{(\text{fixed})} + \mathbf{t}^{(\text{max})\top} \mathbf{r} \\ \mathbf{r} \succeq \mathbf{r}^{(\text{min})} \\ \mathbf{r} \preceq \mathbf{1} \end{array} \quad (4)$$

where $T^{(\text{fixed})} = T^{(\text{init})} + T^{(\text{final})} + \sum_i T_i^{(\text{setup})}$ is the resolution-independent portion of the frame time, $T^{(\text{init})}$ and $T^{(\text{final})}$ are the frame initialization and finalization times, $\mathbf{t}^{(\text{max})}$ is the vector containing the maximum rendering time $T^{(\text{tri})} \cdot maxtri(M)$ for each mesh M, and $\mathbf{r}^{(\text{min})}$ is the vector of minimal resolutions as of equation 3. The constants $T^{(\text{setup})}$, $T^{(\text{tri})}$, $T^{(\text{pix})}$, $T^{(\text{init})}$, and $T^{(\text{final})}$ can be determined by benchmarks in a preprocessing step. As these constants obviously depend on rendering attributes such as shading model, number of light sources, and texturing, we pre-compute their values for each combination of rendering attributes and choose at run-time the appropriate set of values to use for each object.

The cost model presented here assumes an ideal environment in which rendering time is dictated only by rendering operations. In practice, however, process scheduling, page faults, and other direct or indirect blocking kernel calls are out of user control and have an impact on the rendering time. Our current approach to reduce unwanted variations in frame-rate is to add to $T^{(fixed)}$ a worst case estimate of the impact of the system and application environment on the rendering time.

4.2 Benefit heuristics

The $benefit(W, S(\mathbf{r}))$ heuristic provides an estimation of the quality of the image produced when rendering the multiresolution objects in S at resolutions $\mathbf{r}$. We currently use a simple formula derived from [3]:

$$benefit(W, S(\mathbf{r})) = \sum_i importance(S_i, W) accuracy(S_i, r_i) \quad (5)$$

where $importance(M, W)$ is a factor measuring the importance of the object M from the viewpoint W and $accuracy(M, r)$ is a factor measuring how well the mesh at resolution r approximates the mesh at maximum resolution. Currently, we model object importance by

$$\begin{array}{l} importance(M, r) = \\ \quad coverage(M, W) focus(M, W) semantics(M) \end{array} \quad (6)$$

where $coverage(M, W)$ is an estimation of the number of pixels covered by the object, $focus(M, W)$ is the distance of the object's projection to the center of the screen, and $semantics(M)$ is a user-definable object importance factor (e.g. to impose higher quality for interactively manipulated objects). We have experimented with several definition of the accuracy heuristics. Visually pleasing results were obtained using $accuracy(M, r) = \sqrt{N_v^{(\text{max})} r}$ for a mesh with $N_v^{(\text{max})}$ vertices at the highest level of detail, which provides diminishing returns at higher resolutions and, intuitively, relates representation accuracy to an indication of the distance between samples on the surface.

4.3 Temporal coherence

The benefit heuristic defined in the previous section measures image quality and does not depend on its variation over time. It is often useful to include temporal coherence as a quality component to minimize sudden changes in resolution of visible objects, for example when extremely large objects appear or disappear. This can be done by including an hysteresis factor penalizing resolution changes in equation 5. We define hysteresis as follows:

$$hysteresis(M, r) = 1 - \frac{1}{n} \sum_{i=1}^{n} (r_i - r_i^{old})^2 \quad (7)$$

where n is the number of visible objects and $\mathbf{r}^{(old)}$ their resolution at the previous frame. We have found, however, that for most complex scenes hysteresis is not necessary, and this factor is not included in the examples presented in the paper.

5 Optimization algorithm

5.1 Idea of a barrier method

As benefit and cost functions are convex and smooth, a very straightforward method for solving the resolution optimization problem with a guaranteed specified accuracy ϵ is to use an interior point algorithm for convex optimization [27]. The idea of interior point methods is to transform the original problem into an effectively unconstrained problem by incorporating in the objective a barrier function which is smooth, convex, and tends to infinity as the parameter approaches the boundary of the feasible set. Thus, if

an initial feasible point is known, the transformed problem can be efficiently solved using standard unconstrained convex minimization techniques (e.g Newton's method [19]).

In our case, using a logarithmic barrier function, the resolution optimization problem becomes:

$$
\begin{aligned}
\text{Minimize:} \quad & \frac{2n+1}{\epsilon} f_0(\mathbf{x}) + \Phi(\mathbf{x}) \\
\text{Subject to:} \quad & f_i(\mathbf{x}) \leq 0, \qquad i = 1, \ldots, 2n+1, \\
\text{Where:} \quad & x_i = \sqrt{N_v^{(\max)}} r, \qquad \text{i} = 1, \ldots, \text{n} \\
& f_0(\mathbf{x}) = -\sum_i^n importance(S_i, W) x_i \\
& \Phi(\mathbf{x}) = -\sum_{i=1}^{2n+1} \log\left(-f_i(\mathbf{x})\right) \\
& f_1(\mathbf{x}) = T^{(\text{fixed})} + \sum_j t_j^{(\max)} x_j^2 - t^{(\text{desired})} \\
& f_{i+1}(\mathbf{x}) = -x_i + \sqrt{r_i^{(\min)}}, \qquad \text{i} = 1, \ldots, \text{n} \\
& f_{i+1+n}(\mathbf{x}) = x_i - 1, \qquad \text{i} = 1, \ldots, \text{n}
\end{aligned}
\tag{8}
$$

where f_0 is the benefit heuristic of equation 5, $f_1, \ldots, f_{2n+1}$ are functions which are non negative if the cost constraints in equation 4 are met, and Φ is the logarithmic barrier function penalizing constraint violations. It can be demonstrated that the solution of this problem is at a point $\mathbf{x}^\star(\epsilon)$ in which $f_0(\mathbf{x}^\star(\epsilon)) - f_0(\mathbf{x}^\star) \leq \epsilon$, where $\mathbf{x}^\star$ is the optimum solution of the original problem [27].

In other words, while the computation of the optimal resolution for a scene with potentially visible objects implies the solution of a linear problem with 1 quadratic and $2n$ linear constraints, a solution with accuracy ϵ can be found by unconstrained minimization of the smooth convex function $g(\mathbf{x}) = \frac{2n+1}{\epsilon} f_0(\mathbf{x}) + \Phi(\mathbf{x})$.

The generation of a feasible starting point for this problem is straightforward, since we have known lower resolution bounds $\mathbf{r}^{(\min)}$ and a cost heuristic increasing monotonously with $\mathbf{r}$. The simplest solution is to start from $\mathbf{r}^{(\min)} + \epsilon\mathbf{1}$ for a small ϵ. An incremental technique that produces a value which is closer to the optimum is to start from the resolution of the objects computed at the previous frame (or from the minimal resolution for newly visible objects) and to iteratively reduce object resolutions by factor α starting from the objects with the lowest benefit/cost ratio. The iteration terminates when problem becomes feasible or all objects are at the lowest resolution.

5.2 Time-critical sequential unconstrained minimization

Depending on the size of the problem and required accuracy, directly solving equation 8 for a small ϵ may require an excessive number of Newton iterations or fail due to numerical problem. A more efficient way to do it is, instead, to solve a sequence of problems for decreasing values of ϵ, starting at each step from the previously computed sub-optimal value [15]. This nonlinear programming technique is known as *sequential unconstrained minimization*. It is easy to prove that, starting with a requested accuracy $\epsilon^{(0)}$ and updating it at each iteration by a factor $\frac{1}{\mu}, \mu > 1$, the algorithm converges to an accuracy ϵ after exactly $1 + \lceil \log_\mu(\epsilon/\epsilon^{(0)}) \rceil$ steps [15].

In our case, the optimization algorithm itself has to work in a time-critical context, which means that it has to ensure that an approximate result is available at certain time-deadlines. The termination criterion for a time-critical implementation of the algorithm has thus to be the expiration of the allocated time and not only an accuracy criterion.

Algorithm 1 Time-critical rendering optimization

Require: a visible object set $S(\mathbf{r})$ of size n
  Given: timing constraints $t^{(\text{desired})}$, $t^{(\text{optimize})}$
  Given: desired accuracy $\epsilon^{(\text{desired})}$
  Given: machine parameters $T^{(\text{setup})}$, $T^{(\text{tri})}$, $T^{(\text{pix})}$, $T^{(\text{init})}$, $T^{(\text{final})}$
  Given: algorithm parameters $\epsilon^{(0)} > 0, \mu > 1$
  $\mathbf{r}^{(\min)} \leftarrow$ minimum resolution as of equation 3
  $\mathbf{r} \leftarrow$ strictly feasible point as of section 5.1
  $\epsilon \leftarrow \epsilon^{(0)}$
  $\mathbf{r}^\star(\epsilon) \leftarrow \arg\min_{\mathbf{r}}(\frac{2n+1}{\epsilon} f_0(\mathbf{r}) + \Phi)$
  $\mathbf{r} \leftarrow \mathbf{r}^\star(\epsilon)$
  while $\epsilon > \epsilon^{(\text{desired})}$ **or** time elapsed $< t^{(\text{optimize})}$ **do**
    $\epsilon \leftarrow \frac{\epsilon}{\mu}$
    $\mathbf{r}^\star(\epsilon) \leftarrow \arg\min_{\mathbf{r}}(\frac{2n+1}{\epsilon} f_0(\mathbf{r}) + \Phi)$
    $\mathbf{r} \leftarrow \mathbf{r}^\star(\epsilon)$
  end while
Ensure: $\mathbf{r}$ is ϵ-suboptimal
Ensure: $\epsilon \leq \epsilon^{(\text{desired})}$ **or** time elapsed $\approx t^{(\text{optimize})}$
Ensure: rendering time of $S(\mathbf{r})$ will be less than $t^{(\text{desired})}$

This idea is used in algorithm 1. The proposed method has the following properties:

- **controlled optimization and rendering time**: both the CPU time spent in the optimization and the time that will be spent in rendering the meshes at the resolution suggested by the optimization algorithm are parameters of the algorithm and can be externally imposed;
- **customizable quality measure**: the algorithm does not depend on the specific quality measure presented in this paper but only on the fact that the function is convex and smooth;
- **certificate of sub-optimality**: the algorithm not only provides a suboptimal solution but also a bound on the distance in quality from the optimal one. This measure provides a direct indication on whether the resources allocated to the optimization are sufficient which is of great help during the design phase of time-critical programs.

While certain other algorithms share some of these properties, they typically do not meet the capability of our algorithm in all of these areas. Its characteristics make it an ideal candidate for an optimization stage in a time-critical rendering pipeline.

5.3 Time-critical rendering pipeline

A time-critical rendering pipeline aims to display an image at certain time-deadlines independently of the complexity of the scene. To reach this goal, we exploit the properties of our algorithm by adaptively controlling at each frame both the time budget allocated to the optimization and the desired display time (see figure 1).

The parameters under system control are the maximum visual feedback lag $T^{(\text{lag})}$, and the fraction of the frame time to devote to optimization. At each frame, we perform the culling, optimization, and display steps in a sequence. The culling step's time may vary and is dependent on the type of algorithm used and on the complexity of the scene as seen from the current viewpoint. Before starting the optimization step, we measure how much of the frame time is still available and allocate in this range the appropriate time budgets for the optimization and display steps. The optimization step

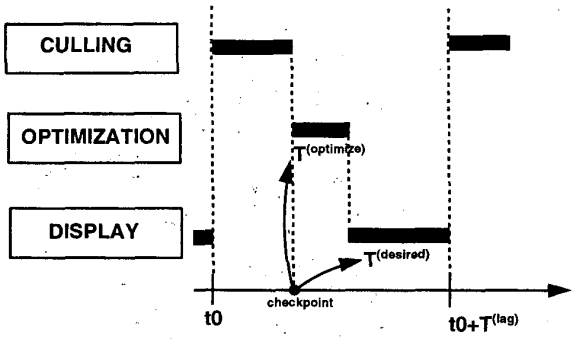

Figure 1: **Time-critical rendering pipeline.** The total frame time is imposed by the user. The time budget allocated to the optimization and display stages is decided after measuring how much time has been spent in the culling stage.

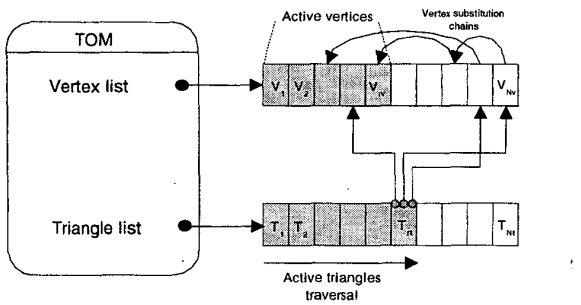

Figure 2: **TOM data structure.** Multiresolution meshes are stored using a vertex list and a triangle list sorted according to contraction resolution.

is run for the allocated time and its result is then passed to the final display stage. This time-critical computing approach bounds the maximum visual feedback lag and enables the use of prediction techniques that extrapolate past user input data to future time points for lag reduction [24].

On a single-processor machine, the maximum visual feedback lag also dictates the maximum visual feedback frequency. On a multi-processor machine, visual feedback frequency can be independently controlled using separate threads for each pipeline stage, as in [20].

6 Multiresolution mesh representation

Our optimization algorithm is independent from the particular data structure used to represent multiresolution meshes. The only requirements are the ability to represent a mesh with an arbitrary number of triangles and to traverse the structure at arbitrary resolutions faster than the graphics pipe-line or, at least, in a time compatible with our linear cost model. An additional requirement for our approach to be practical for large scene databases is data structure compactness.

The Progressive Mesh (PM) [11] representation is a suitable candidate structure. However, the PM representation is compact but cannot be rendered directly, since it has first to be traversed to construct a single resolution mesh structure which is then used for rendering [12]. Managing the dynamic mesh structures associated to each multiresolution representation introduces both time and space overheads in scene rendering application. Experimental results [12] indicate 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 discontinuities in scene complexity caused by objects entering into or exiting from the viewing frustum [3].

In this section, we propose a simple multiresolution triangle mesh structure (TOM: Totally Ordered Mesh) that efficiently supports vertex packing and indexing. The structure is compact, requiring only a small overhead over the single full resolution mesh, and provides fast triangle and vertex traversal rates at any resolution. A similar structure has been independently developed by Guéziec et al. [7] for streaming geometry in VRML.

6.1 TOM: Totally Ordered Mesh

Several algorithms have been recently published that simplify a polygonal mesh by iteratively contracting vertex pairs (e.g. [14, 5, 6, 11, 21, 4]). A vertex pair contraction operation, denoted $(v_1, v_2) \rightarrow \bar{v}$, replaces two vertices (v_1, v_2) with a single target point $\bar{v}$ 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 contracted are chosen and in where the new vertices are positioned. We define vertex substitution, denoted $v_1 \rightarrow v_2$, the restricted form of vertex pair contraction where the target point $\bar{v}$ is constrained to be the second vertex of the pair (v_2). By iteratively applying vertex substitution, a triangle mesh can be reduced by removing one vertex and possibly some degenerated faces at a time. Recent research results demonstrate that good simplification quality and speed can be obtained using this technique [1].

As iterative vertex substitution does not modify vertex data and does not add new vertices, the only information that has to be stored explicitly is the vertex substitution history of each vertex. 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 (see figure 2).

By ordering the vertex list, we obtain a packed representation where the active vertices at vertex resolution $r_v = \frac{n}{N_v}$ are exactly the first n ones in the vertex array of size N_v. 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.

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 vertex resolution at which the transformation occurs and the reference to the vertex by which it is to be substituted. As vertices are sorted according to their resolution, only the vertex reference needs to actually be stored, since the vertex resolution is implicit in the vertex index. The minimal vertex resolution of a triangle, i.e. the vertex resolution at which a triangle is removed from the mesh because of the contraction of one of its edges does not need to be stored, as it can be retrieved in a short time by traversing the substitution chains of its vertices.

To render a mesh with a specified number of triangles n, we first determine the minimal vertex resolution of triangle number n, then traverse the first n triangles following the vertex chains until the active vertices are reached. The lengths of these chains are limited and relatively independent from the model size. In any case the depth at full resolution is always 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. With this representation, the space overhead over a single-resolution mesh representation is equal to just one vertex index per vertex. For a typical mesh of $N_t = 2N_v$ triangles, considering to use a single word to represent both a vertex index and a floating point number, the overhead associated to the above structure is of about 8% of the single full resolution mesh memory footprint when only position and normal are associated to vertices and becomes smaller if other attributes such as colors and texture coordinates are present.

7 Implementation and results

An experimental software library supporting the time-critical multiresolution scene rendering algorithm described in this paper has been implemented and tested on Silicon Graphics IRIX and Windows NT machines. The results presented here were obtained on a Silicon Graphics 320 PC running Windows NT 4.0 and configured with a single 500 MHz Pentium III processor with 512 Kb L2 cache, 256 Mb RAM, and a Cobalt graphics chipset.

To test the behavior of our algorithm, we have written a simple walkthrough application on top of our multiresolution modeling and time-critical rendering libraries. In this application, the culling phase uses a simple bounding box test, the optimization phase uses the algorithm presented in this paper, and rendering is performed in OpenGL, with one positional light, one material per object, Gouraud shading, and one normal per vertex. The application is single-threaded and the high resolution `QueryPerformanceCounter` API is used for all timing measures.

7.1 Cost model coefficients

The cost model coefficients corresponding to the rendering environment used in the benchmark application where determined experimentally by rendering sample objects with a variety of sizes and LODs. Table 1 summarizes the values used for the tests.

Description	Cost model coeff.	Value
Initialization/finalization	$T^{(\text{setup})} + T^{(\text{final})}$	$13889\mu s$
Triangle draw	$T^{(\text{tri})}$	$1.35\mu s/tri$
Pixel fill	$T^{(\text{pix})}$	$0.06\mu s/pix$
Material setup	$T^{(\text{init})}$	$232\mu s/obj$

Table 1: **Cost model coefficient for benchmark application** Experimental measures on a SGI 320, single 500 MHz Pentium III with 512 Kb L2 cache, 256 Mb RAM, Cobalt graphics. Rendering environment: one positional light, one material per object, Gouraud shading, independent triangles, one normal per vertex

7.2 Test environment

We have recorded various parameters during the walkthrough of a test scene containing 166 objects for a total of 1,393,072 polygons

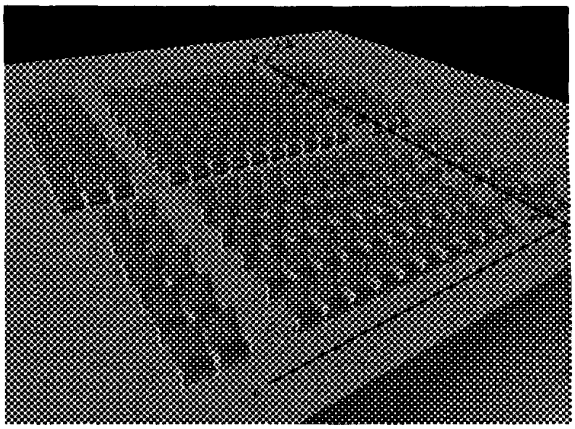

Figure 3: **Scene walkthrough environment**. Test scene at full resolution contains 166 objects for a total of 1,393,072 triangles. The path of the camera is defined by the segments [A,B,C,D]

(see figure 3 as well as figure 6 in color plates). Images were displayed on a 512x512 window.

The camera path has been established so as to include various extreme viewing configurations. All polygons are initially visible from Point A and are progressively exiting from the viewing frustum until point C is reached. After Point C, as the camera is suddenly changing orientation, a large number of objects becomes immediately visible.

Without resolution adaptation, rendering times on the machine used for the tests varies from 50 to 1950 milliseconds per frame depending on the number of visible objects.

7.3 Experimental results and discussion

The number of potentially visible triangles at each frame, as well as the number of triangles actually rendered to meet a display time constraint of $t^{(\text{desired})} = 100ms$ is presented in figure 4. Figure 5 shows frame time statistics for each observer viewpoint along the test path. The predicted frame time closely matches the actual measured frame time, validating our cost model assumptions.

The actual frame time is maintained below the desired time even in the presence of large variations in visual complexity. Even with a relatively large number of objects, we can see that the optimization time remains relatively small compared to the display time. The tests have been performed using a time constraint for the optimization step of $t^{(opt)} = 20ms$ and a target quality accuracy of 5 %. For large portions of the path, this accuracy has been reached in a time sensibly inferior to the allocated limit, leaving more time for other system tasks. In a more elaborate implementation, a feedback algorithm could be used for the adaptation of $t^{(opt)}$.

Figure 7 in the color plate section presents the scene from two viewpoints. In the first row, the objects are rendered in flat shading without any resolution adjustment. The middle row images have been recorded in flat shading using our resolution optimization algorithm. The last row of images has been recorded in the same conditions as the middle row, but depicts the resolution chosen for each object, darker shades of gray representing more detail. The mapping illustrates the effect of the benefit heuristic on the distribution of the polygon budget.

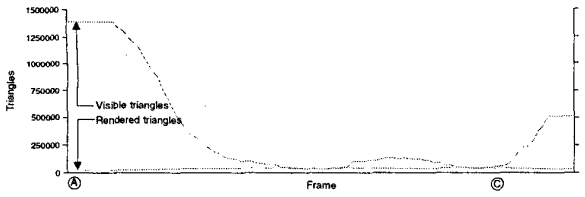

Figure 4: **Visible vs. displayed polygons during the test scene walkthrough.** All polygons are initially visible from Point A and are progressively exiting from the viewing frustum until point C is reached. After Point C, as the camera is suddenly changing orientation, a large number of objects becomes immediately visible.

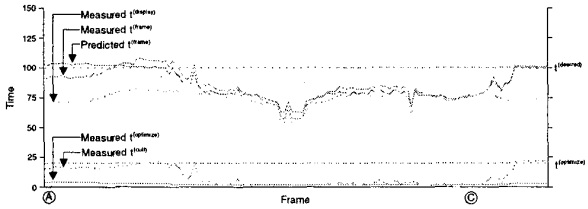

Figure 5: **Frame time statistics for each observer viewpoint along the test path.** Predicted time closely matches the actual frame time, which is always maintained below the targeted $100ms$.

8 Conclusions and future work

We have described a framework for time-critical rendering of graphics scenes with a complex geometric description. Our technique relies upon a scene description in which objects are represented as multiresolution meshes. We perform a constrained optimization at each frame to choose the resolution of each potentially visible object that generates the best quality image while meeting timing constraints. The technique provides smooth level-of-detail control and aims at guaranteeing a uniform, bounded frame rate even for widely changing viewing conditions. Furthermore, our algorithm provides for each image both a quality measure and a bound on the distance in quality from the optimal one. The typical accuracy of our solutions is less than 5%, which is an order of magnitude better than what can be guaranteed by current combinatorial optimization approaches.

The optimization algorithm is independent from the particular data structure used to represent multiresolution meshes. The only requirements are the ability to represent a mesh with an arbitrary number of triangles and to traverse the structure at an arbitrary resolution in a short predictable time. A data structure satisfying these criteria has been presented in section 6.

Our experimental results demonstrate the feasibility of the discussed optimization approach for time-critical rendering on a PC of scenes composed of hundreds of objects and millions of triangles. Our current work is concentrating on extending our approach to per-object view-dependent LODs using a two-step optimization process and on exploiting the possibilities offered by the general convex optimization framework for improving the benefit heuristics with the incorporation of a specific perceptual component. In parallel, we are exploring simpler and possibly faster optimization techniques that exploit the particular structure of the optimization problem associated with the current simplified heuristics. Finally, we plan to explore feedback techniques for dynamically allocating a time budget for the time-critical sequential optimization procedure.

Acknowledgments

The authors would like to thank the anonymous reviewers for their many helpful comments and suggestions.

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] 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.

[3] 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).

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] HONG, L., MURAKI, S., KAUFMAN, A., BARTZ, D., AND HE, T. Virtual voyage: Interactive navigation in the human colon. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), T. Whitted, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 27–34. ISBN 0-89791-896-7.

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

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

[13] HOPPE, H. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings IEEE Visualization'98* (1998), IEEE, pp. 35–42.

[14] 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.

[15] LOOTSMA, F. *Numerical Methods for Non-Linear Optimization.* Academic Press, 1972.

[16] 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.

[17] MASON, A. E. W., AND BLAKE, E. H. Automatic hierarchical level of detail optimization in computer animation. *Computer Graphics Forum 16*, 3 (Aug. 1997), 191–200. Proceedings of Eurographics '97. ISSN 1067-7055.

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

[19] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. *Numerical Recipes*, second ed. Cambridge University Press, Cambridge, 1992.

[20] 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.

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

[22] 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.

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

[24] WLOKA, M. Lag in multiprocessor virtual reality. *Presence: Teleoperators and Virtual Environments 4*, 1 (Sept. 1995), 50–63.

[25] 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.

[26] 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.

[27] YE, Y. *Interior Point Algorithms: Theory and Analysis.* Wiley-Interscience series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1997.

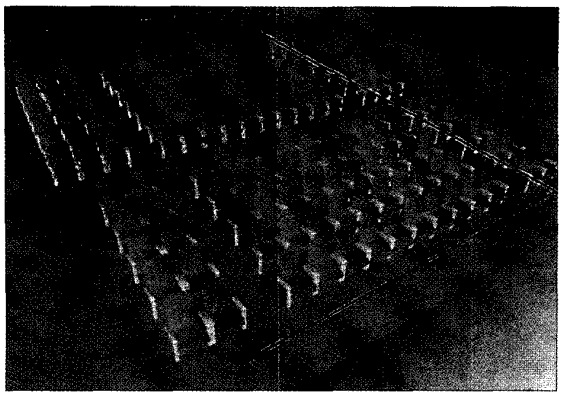

Figure 6: [COLOR PLATE] Test scene at full resolution contains 166 objects for a total of 1,393,072 triangles. The path of the camera is defined by the segments [A,B,C,D]

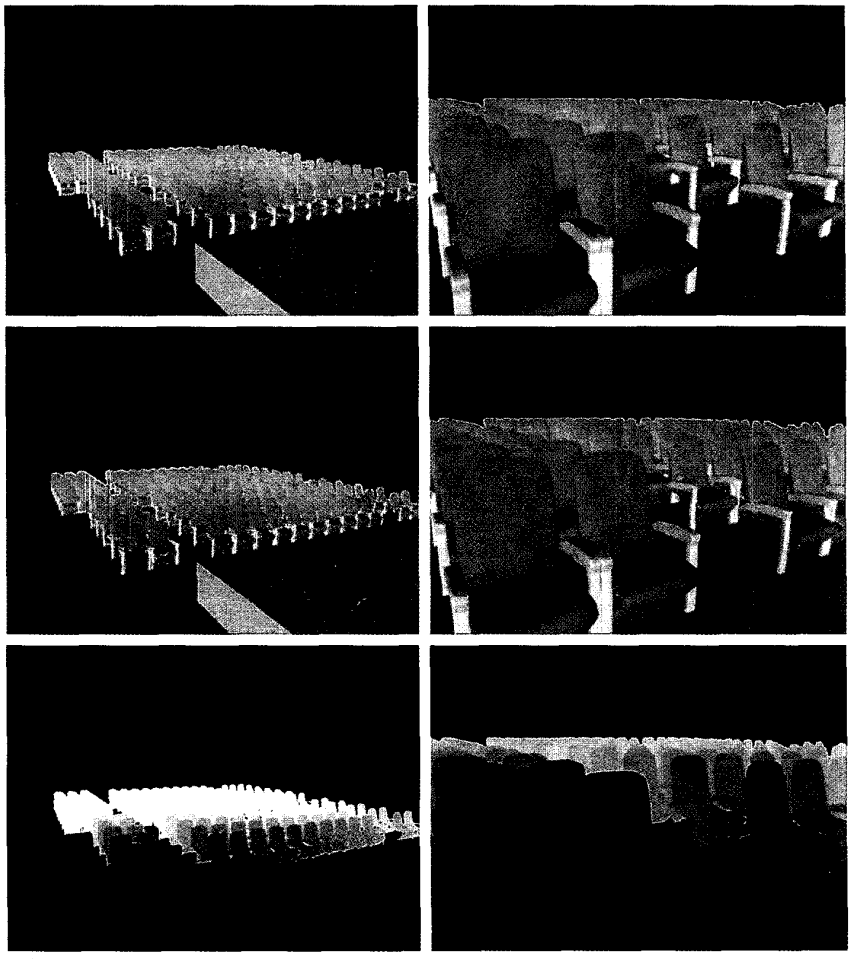

Figure 7: [COLOR PLATE] Images rendered from two points of view. Top images contain full resolution objects. Objects in middle row images have adaptive resolution and are rendered in flat shading. The last row of images has been recorded in the same conditions as the middle row, but depicts the resolution chosen for each object, darker shades of gray representing more detail.