Interactive Multiresolution Surface Viewing

Andrew Certain[‡] *Jovan Popović* *Tony DeRose* *Tom Duchamp*[*]
David Salesin *Werner Stuetzle*[†]

Department of Computer Science and Engineering
[*]Department of Mathematics
[†]Department of Statistics
University of Washington

Abstract

Multiresolution analysis has been proposed as a basic tool supporting compression, progressive transmission, and level-of-detail control of complex meshes in a unified and theoretically sound way.

We extend previous work on multiresolution analysis of meshes in two ways. First, we show how to perform multiresolution analysis of colored meshes by separately analyzing shape and color. Second, we describe efficient algorithms and data structures that allow us to incrementally construct lower resolution approximations to colored meshes from the geometry and color wavelet coefficients at interactive rates. We have integrated these algorithms in a prototype mesh viewer that supports progressive transmission, dynamic display at a constant frame rate independent of machine characteristics and load, and interactive choice of tradeoff between the amount of detail in geometry and color. The viewer operates as a helper application to Netscape, and can therefore be used to rapidly browse and display complex geometric models stored on the World Wide Web.

CR Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling — surfaces and object representations; J.6 [Computer-Aided Engineering]: Computer-Aided Design (CAD).

Additional Keywords: Geometric modeling, wavelets, multiresolution analysis, texture mapping, viewer.

1 Introduction

Three-dimensional meshes of large complexity are rapidly becoming commonplace. Laser scanning systems, for example, routinely produce geometric models with hundreds of thousands of vertices, each of which may contain additional information, such as color.

Working with such complex meshes poses a number of problems. They require a large amount of storage and consequently are slow to transmit. Additionally, they contain more faces than can be interactively displayed on any current hardware.

Existing viewers either do not deal with these problems at all, or do so only in crude ways, for example by showing wireframes or by displaying only a fraction of the faces during dynamic viewing, and then switching back to surfaces once the motion has stopped.

[‡]Department of Computer Science and Engineering,
University of Washington, Box 352350, Seattle, WA 98195-2350

A more sophisticated way of coping with both the transmission and dynamic display problems is to use a precomputed sequence of lower detail approximations to the mesh. Such approximations can be computed, for example, using the method of Rossignac and Borrel [9]. During transmission, a cruder approximation is displayed while the next more detailed approximation is received. For dynamic display, one chooses the highest detail approximation compatible with a desired frame rate.

A major disadvantage of this approach is that the total amount of data that has to be transmitted and stored is larger than the description of the full resolution mesh. In fact, there is a tradeoff between granularity (the difference in resolution between successive models) on the one hand and transmission time and storage requirements on the other hand.

Previous work [1, 5, 6, 10] has demonstrated that, at least in principle, multiresolution analysis offers a unified and theoretically sound way of dealing with these problems. A multiresolution representation of a mesh consists of a simple approximation called the base mesh, together with a sequence of correction terms called wavelet coefficients which supply the missing detail. The key point is that truncated sequences of wavelet coefficients define approximations to the mesh with fewer faces.

Although promising, previous work is lacking in at least two ways. First, either color or geometry were represented in multiresolution form, but not both. Second, algorithms for reconstructing and displaying multiresolution meshes were much too slow for interactive use.

In this paper we address both of these deficiencies. We deal with complex colored meshes using separate multiresolution representations for geometry and color that are combined only at display time. We also describe efficient algorithms and data structures that allow us to incrementally construct and render lower resolution approximations to the mesh from the color and geometry wavelet coefficients at interactive rates.

The separation of color and geometry, together with our incremental algorithms, allows the efficient implementation of the following features:

- ***Progressive transmission:*** We first transmit and display the base mesh and then transmit the wavelet coefficients in decreasing order of magnitude. As wavelets are received, they are incorporated into the approximation, and the approximation is periodically re-rendered. In the examples we have tried the approximation rapidly converges to the original mesh (see Color Plates 1(a–d)). Only a small penalty is incurred for progressive transmission (see Section 5.1).
- ***Performance tuning:*** By truncating the color and geometry expansions we can obtain lower detail approximations of the mesh with essentially any desired number of faces. During dynamic

display, we truncate the expansions at a level of detail that can be rendered with the desired frame rate. We monitor the frame rate and dynamically modify the level of detail in response to changing machine load.

- ***Automatic texture map generation:*** The separation between color and geometry and the way in which they are represented allows us to take advantage of texture-mapping hardware, as described in Section 3.3. Color Plates 1(g) and 1(h) illustrate the gains obtained by exploiting texture mapping. For a given number of polygons, texture mapping allows display of a far better approximation (Color Plate 1(h)), as all the polygons can be dedicated to capturing geometric detail. Color can always be displayed at full resolution because adding color detail does not increase the polygon count.
- ***Adapting to user preferences:*** Color and geometry expansions can be truncated independently. In the absence of texture mapping, the number of faces of the resulting mesh will depend on the truncation thresholds. There will in general be many combinations of color threshold and geometry threshold that result in approximately the same number of faces (see Color Plates 1(e–g)). Automatically finding the combination giving the "best looking" approximation seems to be a hard problem, as it will certainly depend on the model itself. Instead, we allow the user to interactively choose the tradeoff.

To demonstrate our ideas, we have built a prototype viewer running as a helper application for Netscape. As demonstrated in the accompanying videotape, our viewer can be used to rapidly browse and display complex geometric models stored on the World Wide Web.

The rest of the paper is organized as follows. In Section 2 we present a brief summary of multiresolution analysis of colored meshes. In Section 3 we describe the basic data structures and algorithms for efficiently constructing and rendering truncated models. In Section 4 we sketch the architecture of our viewer. In Section 5 we present the results of several numerical experiments. Finally, Section 6 contains a discussion and ideas for future work.

2 Background

In this section we first present a synopsis of multiresolution analysis for piecewise linear functions on triangular meshes. For a more complete exposition, see Stollnitz *et al.* [11]. We then describe how to convert an arbitrary colored mesh to a parametric form amenable to multiresolution analysis.

2.1 Multiresolution analysis

The central idea of multiresolution analysis is to decompose a function into a low resolution ("coarse") part and a sequence of correction ("detail") terms at increasing resolutions. Multiresolution analysis for functions on $\mathbf{R}^n$ was formalized by Meyer [8] and Mallat [7]. Lounsbery [5] and Lounsbery *et al.* [6] extended multiresolution analysis to a class of functions including functions defined on triangular meshes, which we call *level J piecewise linear.* A function f defined on a triangular mesh M^0 is called level J piecewise linear if it is piecewise linear on the mesh M^J obtained by performing J recursive 4-to-1 subdivisions of the faces of M^0 (see Figure 1).

Let $\hat{V}^j$ denote the vector space of level j piecewise linear functions on M^0. Let $\hat{\phi}_i^j$ denote the unique level j piecewise linear function assuming value 1 at vertex i and value 0 at all other vertices of M^j. These *level j hat functions* form a basis of $\hat{V}^j$. In the context of multiresolution analysis they are often referred to as *scaling functions.* The spaces $\hat{V}^0, \hat{V}^1, \ldots$ form a nested sequence, as required by multiresolution analysis.

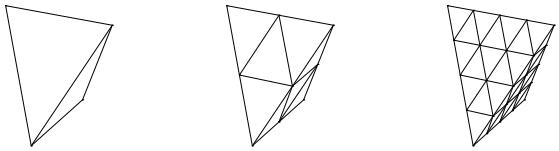


Figure 1 Recursive 4-to-1 splitting of a tetrahedron: (a)M^0, (b) M^1, (c) M^2.

Besides a nested sequence of spaces, the other basic ingredient of multiresolution analysis is an inner product. We use the inner product

$$\langle f \mid g \rangle = \sum_T \int_{\mathbf{x} \in T} f(\mathbf{x})\, g(\mathbf{x})\, d\mathbf{x},$$

where the sum is taken over all faces of M^0 and $d\mathbf{x}$ is the area element, normalized so that all faces of M^0 have unit area.

Given a nested sequence of function spaces and an inner product, we can now define wavelets. The orthogonal complements $\hat{W}^j$ of $\hat{V}^j$ in $\hat{V}^{j+1}$, for $0 \le j < J$, are called orthogonal *wavelet spaces.* A *wavelet basis* for $\hat{V}^J$ consists of the level 0 scaling functions, together with bases for the wavelet spaces $\hat{W}^0, \ldots, \hat{W}^{J-1}$. Given such a wavelet basis, we can express any level J piecewise linear function f on M^0 as a linear combination of scaling functions and wavelets at various levels.

Ideally we would like the wavelets, together with the level 0 scaling functions, to form an orthonormal basis for $\hat{V}^J$. We could then calculate the best k term L^2 approximation to a function $f \in \hat{V}^J$ by keeping the k terms of the expansion with the largest coefficients. On the other hand, we want wavelets to have small support so that the contribution to the approximation from each wavelet term can be rapidly incorporated into the model. Unfortunately, orthogonality of wavelet spaces and small spatial support of wavelets are conflicting goals. As small spatial support is essential for applications, we relax the orthogonality requirement.

Lounsbery *et al.* [6] stipulate *a priori* the size k of the support and then construct *biorthogonal wavelets* $\hat{\psi}_i^j$ that span $\hat{W}^j$ and are as orthogonal as possible to $\hat{V}^j$. The wavelets obtained in this way are called *k-disk wavelets* [11].

More precisely, consider a vertex i of M^{j+1} that is located at the midpoint of an edge e of M^j. The *k-disk wavelet* centered at vertex i is a function of the form

$$\hat{\psi}_i^j = \hat{\phi}_i^{j+1} + \sum_{v \in N_k} s_v^j \hat{\phi}_v^j, \tag{1}$$

where N_k denotes a set of level j vertices in a neighborhood of vertex i. The neighborhoods N_k are defined recursively. The neighborhood N_0 for the 0-disk wavelet consists of the endpoints of e; the neighborhood N_k contains the vertices of all triangles incident on N_{k-1} (see Figure 2). The wavelet consisting of only the level $j+1$ scaling function is called the *lazy wavelet.*

The coefficients s_{iv}^j are chosen to minimize the norm of the orthogonal projection of $\hat{\psi}_i^j$ onto $\hat{V}^j$. They are determined by solving the following system of linear equations:

$$\sum_{v \in N_k} \langle \hat{\phi}_u^j \mid \hat{\phi}_v^j \rangle\, s_{iv}^j = -\langle \hat{\phi}_u^j \mid \hat{\phi}_i^{j+1} \rangle, \quad \text{for all } u \in N_k.$$

Note that the system is local to vertex i. The size of the system for 0-disk wavelets is only 2×2. For larger values of k the size of the

system depends on the valence of the parent vertices; in regular regions of the mesh where all vertices have valence 6, the system has size 10×10 for $k = 1$, and size 24×24 for $k = 2$.

The process of expressing a level J piecewise linear function in terms of level 0 scaling functions and wavelets is called *filterbank analysis.* For a description see Stollnitz *et al.* [11].

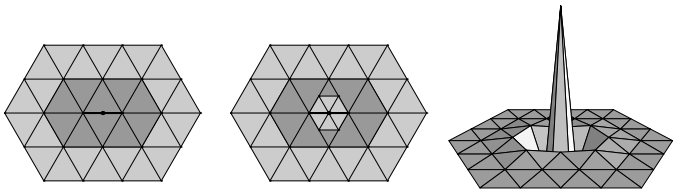


Figure 2 (a) The support of the 1-disk wavelet $\hat{\psi}_i^j$. Dark shaded area: N_0-neighborhood of center edge; light shaded area: faces added to form N_1-neighborhood. (b) The triangles required to introduce $\hat{\psi}_i^j$ during reconstruction. (c) The graph of $\hat{\psi}_i^j$.

2.2 Conversion of colored meshes to multiresolution form

Multiresolution analysis of a colored mesh M is based on the premise that M is defined parametrically by two vector valued level J piecewise linear functions, a *geometry function* f_{geom} and a *color function* f_{color}, each mapping a triangular *base mesh* M^0 into $\mathbf{R}^3$.

Typically, M will not be given in this form, but instead in the form of vertices, edges, and faces, vertex positions, and vertex colors. In order to apply multiresolution analysis, M must be converted to parametric form. We do this by first applying the remeshing algorithm of Eck *et al.* [1]. The output of the remeshing algorithm is a base mesh M^0 with a relatively small number of faces, a parameterization $\rho : M^0 \rightarrow M$, and an approximation of ρ by a level J piecewise linear embedding $f_{geom} : M^0 \rightarrow \mathbf{R}^3$ of the form $f_{geom} = \sum_i f_i \hat{\phi}_i^J$, where f_i are vectors in $\mathbf{R}^3$ representing the geometric positions of the vertices of M^J.

We next apply the filter bank analysis algorithm of Lounsbery *et al.* [6] to obtain a wavelet expansion of f_{geom}. Note that this analysis will generate a vector of three coefficients for each wavelet, one for each of the three coordinate functions. We sort these coefficient vectors in order of decreasing length and then store them together with identifiers for the wavelets (center vertex and level) in a file called the *geometry-wavelet file.*

We now turn to multiresolution analysis of color. Color is originally given at the vertices of M, and can be extended to all of M by linear interpolation. The parametrization $\rho : M^0 \rightarrow M$ obtained during remeshing induces a color function γ on M^0. To construct a level J piecewise linear approximation f_{color} to γ, we sample γ at the vertices of M^J. As in the case of geometry, we then compute the wavelet expansion of f_{color} by filterbank analysis and store the wavelet coefficient vectors in order of decreasing length in a *color-wavelet file.*

The base mesh M^0, its vertex positions (the coefficients of the level 0 scaling functions in the expansion of f_{geom}) and its vertex colors are stored in a *base file.* The geometry-wavelet file, the color-wavelet file, and the base file constitute the input to our multiresolution viewer.

3 Algorithms and data structures

In this section we describe the algorithms and data structures that form the basis of our multiresolution viewer. We assume that the colored mesh is represented in multiresolution form, i.e., by a base mesh and wavelet expansions of the color and geometry functions.

At the full resolution, the number of faces of M^J is 4^J times the number of faces of M^0. The faces of M^J can be naturally organized into a tree Q. The root of Q has as many children as there are faces in the base mesh, while every other internal node has four children. Each leaf of Q corresponds to a face of M^J. This tree organization was also used by Schroeder and Sweldens [10]. The mesh is rendered by traversing the tree Q, evaluating f_{geom} and f_{color} at the vertices, and generating a colored triangle for each leaf.

In the absence of texture-mapping hardware, color and geometry are handled identically, so we will couch the discussion in terms of geometry alone. The use of texture mapping is the topic of Section 3.3.

First some terminology: let f_{geom}^r denote the approximation to f_{geom} obtained by summing the scaling functions and the largest r wavelets, and let Q^r denote the smallest subtree of Q we for which f_{geom}^r is linear on each leaf.

For progressive transmission we first transmit the base mesh M^0 and the coefficients of the level 0 scaling functions. The associated tree Q^0 consists only of the root node and as many leaves as there are faces in the base mesh. As wavelets arrive, we incrementally grow Q^r and update f_{geom}^r, and periodically render the mesh.

Use of the wavelet representation for performance tuned viewing and level-of-detail control is based on the observation that for small r, the tree Q^r will also be small, and therefore rendering Q^r will result in many fewer triangles than rendering Q. In principle we could generate approximations with almost any desired number of faces by growing from the base mesh. For efficiency reasons we cache trees and vertex positions for a sequence of approximations, and then grow the desired tree from the closest approximation with fewer than the desired number of faces.

3.1 Data structures

As previously stated, the primary data structure used to represent the mesh is a tree Q, which has as many descendents from the root as there are faces in the base mesh and is a quadtree for all other levels.

We represent all nodes of Q, except for the root, with the following data structure:

```
type Face = record
    level: Integer
    children[4]: array of pointer to Face
    cornerVertex[3]: array of pointer to Vertex
    edgeVertex[3]: array of pointer to Vertex
end record
```

A face is said to be *of level j* if it is a face of M^j. The array *cornerVertex* has pointers to three vertices of the face, and the array *edgeVertex* has pointers to three vertices that subdivide the edges of this face.

We represent vertices with the following data structure:

```
type Vertex = record
    parentV[2]: array of pointer to Vertex
    parentF[2]: array of pointer to Face
    fGeom: XYZposition
    fColor: RGBcolor
    g: XYZvector
    hGeom[ ]: array of HatFunctionCoefficients
    hColor[ ]: array of HatFunctionCoefficients
end record
```

The array *parentV* contains pointers to the two vertices on either end of the edge that the vertex subdivides — these are called *parent vertices* of the vertex. The array *parentF* contains pointers to the two

faces on either side of the edge that the vertex subdivides — these are called the *parent faces* of the vertex. A vertex is said to be *of level j* if it was created at the j-th level of subdivision, i.e., if its parent faces are of level $j-1$. The fields *fGeom* and *fColor* contain the values of f^r_{geom} and f^r_{color} at the vertex. The role of *hGeom* and *hColor* is explained in Section 3.2.

Vertices of level $j > 0$ are indexed by the base face they lie in, together with their barycentric coordinates within the face. As it is often necessary to find the node representing a vertex from its index, we maintain an auxiliary hash table that maps vertex indices to vertex nodes. Whenever a vertex is created, it is added to the table.

3.2 Algorithms

Suppose we have already constructed the face tree Q^r and evaluated f^r_{geom} at all its vertices. Adding a wavelet requires growing Q^r into Q^{r+1} and evaluating f^{r+1}_{geom}. For efficiency reasons we do not re-evaluate f^{r+1}_{geom} for every new wavelet. Instead we *gather* a sequence of s wavelets, then *evaluate* f^{r+s}_{geom} when the new mesh is rendered. We now describe the gather and evaluate stages.

3.2.1 The gather stage

Gathering a wavelet $\hat{\psi}^j_i$ with wavelet coefficient a^j_i involves three steps:

1. Decompose the term $\hat{\psi}^j_i$ into a sum of hat functions at level j and $j+1$ according to Equation (1).
2. For each hat function in the decomposition, grow the current face tree to accommodate it. A face tree is said to *accommodate* a function if the function is linear over each face. This process is described more fully below.
3. For each hat function in the decomposition $\hat{\phi}^{j'}_v$, $j' = j, j+1$, centered at vertex v, update the *hGeom* field of v: $v.hGeom[j'] \mathrel{+}= a^j_i s^{j'}_v$, where $s^{j'}_v$ is the coefficient of $\hat{\phi}^{j'}_v$ in the decomposition of step 1.

The most complicated part of gathering is growing the current face tree Q^r to accommodate a level j hat function $\hat{\phi}^j_v$ centered at a vertex v. We call a level j vertex *complete* if its parent faces have been subdivided. (By definition, all level 0 vertices are complete.) As each vertex has pointers to its two parent faces (nil if a parent face does not exist), it is easy to test a vertex for completeness.

Clearly, Q^r can accommodate a hat function $\hat{\phi}^j_v$ if the level j neighbors of vertex v are complete. Thus, there is a simple recursive procedure to make a vertex complete:

- Make its two parent vertices complete;
- Subdivide the two parent faces of the vertex.

Whenever a new vertex w is created in the completion process, f^r_{geom} is evaluated at the vertex, and the value is recorded in *w.fGeom*. Since f^r_{geom} is linear on the edges of Q^r, this evaluation is accomplished by averaging the *fGeom* values of w's parent vertices.

While this growing process is simple, it can generate more than the minimum number of triangles needed to accommodate a hat function (see Figure 3).

3.2.2 The evaluate stage

Recall that wavelets are added in two stages. In the *gather* stage the face tree is grown so that it contains all the faces necessary to accommodate the new wavelets. At this stage we also compute the values of the *current* approximation f^r_{geom} at the newly introduced vertex positions. The wavelets are decomposed into hat functions, and the coefficient arrays for their center vertices are updated. The new geometry function f^{r+s}_{geom} is not evaluated until the tree is rendered, at which time the contributions from all the hat functions are summed in a single tree traversal. We will now describe this evaluation stage.

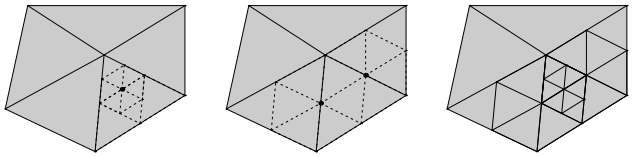


Figure 3 Making a vertex complete: (a) A vertex to be made complete. (The dashed faces are the minimal number that must be added to make the vertex complete.) (b) The parent vertices are created and made complete by subdividing their parent faces. (c) Subdividing the parent faces of the vertex makes it complete.

Let g denote the sum of all the hat functions gathered since the last evaluation stage, and let g_k denote the partial sum obtained by adding all the contributions from hat functions of level k or smaller. By construction $g = g_L$, where L is the maximum level of any leaf of Q^{r+s}. Note that since g_k is linear over the faces of level k and above, it is completely determined by its values at the vertices of Q^{r+s} of level k and less.

We now present an inductive procedure to compute the values of g_L at all of the vertices of Q^{r+s}.

It is easy to compute the values of g_0 at the level 0 vertices — they are the coefficients of the gathered level 0 hat functions.

Next we describe how to compute the values of g_{k+1} at all vertices of Q^{r+s} of level $k+1$ and smaller from the values of g_k at all vertices of level k and smaller. Let h^{k+1}_v denote the coefficient of the level $k+1$ hat function centered at v. If v is a vertex of level k or less, then $g_{k+1}(v) = g_k(v) + h^{k+1}_v$. If v is a level $k+1$ vertex, then it splits an edge connecting its two level k parent vertices, Therefore, $g_k(v)$ is the average of the values of g_k at its parent vertices, and $g_{k+1}(v) = g_k(v) + h^{k+1}_v$.

The calculation of g_L can be performed efficiently during a breadth first traversal of Q^{r+s}, as summarized in the pseudocode given in Figure 4.

3.3 Treatment of color

As mentioned earlier, in the absence of texture-mapping hardware color and geometry are handled identically: both color and geometry wavelets are gathered and evaluated as described in the previous section.

Representing colored meshes in multiresolution form makes it easy to exploit texture mapping hardware. The basic idea is to associate a region of texture memory with each face of the base mesh. If the full resolution model is subdivided to level J, a $2^J \times 2^J$ texture map is allocated, but only the lower diagonal is actually used. (To reduce the wasted texture memory, we pair adjacent base mesh faces whenever possible. We then allocate a square region of texture memory to the pair.)

Since geometry is represented parametrically by a piecewise linear function over M^J, there is a straightforward solution for the normally difficult problem of generating texture coordinates for arbitrary meshes. The texture coordinates for any vertex are simply the pre-image of the vertex under the parametrization. Therefore, the corner vertices of a base mesh face have texture coordinates (0,0), (1,0), and (0,1), and the texture coordinates for every other vertex are the average of its parents' coordinates. The image displayed in

```
procedure Evaluate()
    queue ← Level 0 faces
    do while queue != empty
        currentFace ← GetFirstFace(queue)
        currentLevel ← currentFace.level
        if IsSubdivided(currentFace) then
            for each cornerVertex v of currentFace do
                v.g += v.hGeom[currentLevel]
                v.hGeom[currentLevel] ← 0
            end for
            for each edgeVertex e of currentFace do
                if e has two parent faces then
                    { e will be visited twice, so add 1/2 per visit}
                    e.g += 0.25 (e.parentV[1].g+e.parentV[2].g)
                else
                    e.g += 0.5 (e.parentV[1].g+e.parentV[2].g)
                end if
            end for
            for each i ∈ 0, 1, 2, 3 do
                Append currentFaces.children[i] to queue
        else
            for each cornerVertex v of currentFace
                v.fGeom += v.g + v.hGeom[currentLevel]
                v.g ← v.hGeom[currentLevel] ← 0
            end for
            AddToDisplayList(currentFace)
        end if
    end while
end procedure
```

Figure 4

Figure 5 illustrates texture mapping. The base mesh has been rendered with only the scaling functions of f_{geom}, but with all of the terms of f_{color}.

The texture map associated with a face of the base mesh is initialized by linearly interpolating between the colors at the vertices of the face (i.e., the level 0 color scaling function coefficients). The texture map is updated as soon as color wavelets are received, essentially by "painting" the wavelet into the texture map. Since the addition of color wavelets does not increase the triangle count, systems with texture-mapping hardware color can always display color at its highest resolution.

4 Viewer Architecture

Our viewer, written in OpenGL and Motif for Silicon Graphics Iris workstations, is configured as a helper application for Netscape. When a multiresolution-surface link is followed, the viewer application opens an HTTP connection for the base mesh file. After receiving the base mesh, the viewer displays it in a graphics window (see Figure 5) and opens two parallel HTTP connections, one for the color wavelets file and one for the geometry wavelets file. As wavelet coefficients are received they are incorporated as described in Section 3, and the model is periodically redisplayed. Color Plates 1(a–d) illustrate a model at various stages of transmission. Assuming a 64Kbs link (ISDN speeds), the images shown represent, from top to bottom, the model after 3 seconds, 17 seconds, 59 seconds, and 180 seconds (the full model).

In standard operation, the quality of the model displayed in the viewer is controlled by the slider labeled *Frame Time*. When the user is rotating or translating the model, the viewer attempts to maintain that frame rate by measuring the polygon performance for the previous frames and predicting the desired model size for the upcoming frame. When there is no interaction, a more refined model is rendered, allowing the user to see more detail. If the refined model takes a significant time to render, the rendering is performed in stages, so that the viewer can check for user events during the rendering. If the user decides to interact with the model while the viewer is drawing a refined model, rendering is aborted, and the system returns to interactivity.

The quality of the model can also be controlled in two other ways: the user can explicitly set either the number of geometry and color wavelets to be added to the base mesh, or the number of polygons to be used in creating the approximation.

If either the frame time or the number of polygons is specified, the tradeoff between color and geometry is controlled with the slider labeled *Color to Geom.* Moving the slider to the left indicates a preference for geometry detail, whereas moving it to the right indicates a preference for color detail. The tradeoff is shown in Color Plates 1(e–g), where (e) corresponds to a strong preference for color, (g) corresponds to a strong preference for geometry, and (f) corresponds to a balance between the two. Each of these model consists of the same number of Gouraud shaded polygons.

The color/geometry slider is only active on machines without texture-mapping hardware. If the machine has texture-mapping hardware, color wavelets do not increase the polygon count, so they are always included. Color Plate 1(h) shows the model that can be displayed for the same polygon budget used in Plates 1(e–g).

Plate	wavelet type	# geom wavelets	# color wavelets	# polys	L^2 error	L^∞ error
(a)	0-disk	770	830	4701	.0961	.3217
(b)	0-disk	4166	4445	22725	.0375	.0949
(c)	0-disk	14350	14605	56418	.0076	.0136
(d)	0-disk	49530	49530	98304	2.3e-6	1.92e-6
(e)	0-disk	114	811	3006	.2555	.5246
(f)	0-disk	371	567	3033	.1607	.3461
(g)	0-disk	743	324	3015	.1225	.2777
(h)	0-disk	774	49530	2994	.1203	.2777
(i)	Lazy	16380	16380	32760	6.7e-8	4.7e-7
(j)	Lazy	1254	1350	5561	.0099	.0503
(k)	0-disk	1129	1084	5510	.0075	.0459
(l)	2-disk	735	883	5573	.0092	.0676

Table 1 Statistics for Color Plate 1

5 Results

In this section we present various statistics for the color plates, and we describe a number of numerical experiments we have performed.

Statistics for the color plates are summarized in Table 1. Three different types of wavelets were used as indicated by the second column. All examples were computed using the same type of wavelet for both color and geometry, although in principle different types of wavelets could be used. The other columns should be self-explanatory. The errors reported in the last two columns are normalized so that the crudest model has error 1.

In addition to using the viewer to create the color plates, we conducted a set of numerical experiments to compare the performance of four types of wavelets: lazy, 0-, 1-, and 2-disk wavelets. The experiments focused on the following factors:

- ***Convergence as a function of number of wavelet coefficients:*** For fixed network bandwidth, the rate at which the transmitted model approaches the original depends on how quickly the error decreases as a function of the number of wavelet coefficients.

  Figure 6 is a plot of L^2 error in geometry vs. number of coefficients for the various types of wavelets for the head model shown in Color Plates 1(e–h). The plot of L^∞ error is qualitatively similar. Similar results were obtained for the other two models.

  Our conclusion is that lazy wavelets perform slightly worse than

Figure 5 The multiresolution viewer.

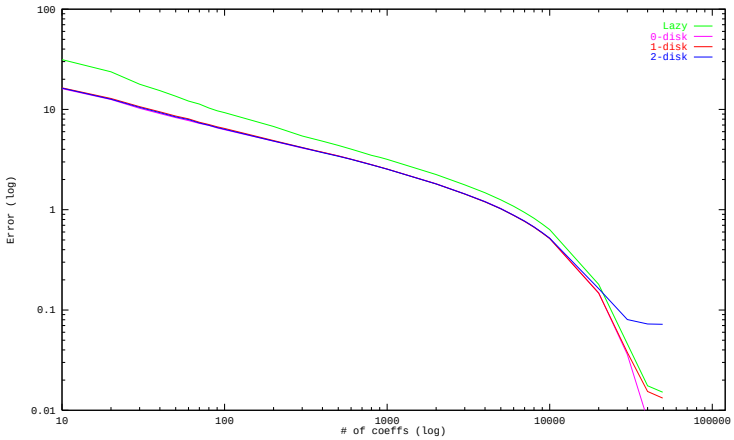


Figure 6 L^2 error vs. number of wavelet coefficients.

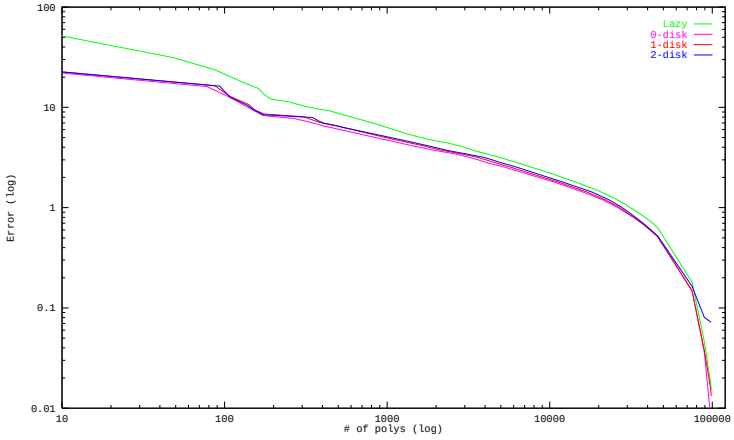


Figure 7 L^2 error vs. number of polygons.

k-disk wavelets, but there seems to be no significant difference between various values of *k*.

- ***Convergence as a function of number of polygons:*** For fixed polygon display rate and update frequency, the visual appearance of the model depends on how quickly the error decreases as a function of the number of polygons in the model.

  Figure 7 is a plot of the L^2 error vs. number of polygons for the same head model. Again, the corresponding plot for the L^∞ error is qualitatively similar.

  Color Plates 1(i–l) illustrate the visual fidelity for the earth model when different types of wavelets are used to produce a model with a fixed polygon count. Table 5 indicates that the error for this number of polygons is actually less for lazy wavelets than for 2-disk wavelets, due to the large number of polygons that a 2-disk wavelet may introduce. Color plate 1(i) is the full-resolution earth model, subdivided to level-6. The next three color plates, 1(j–l), depict the earth reconstructed to approximately 5500 polygons using lazy wavelets, 1(j), 0-disk wavelets, 1(k), and 2-disk wavelets, 1(l). Although there are visual differences between the images, it is not clear which is preferable.

  Our conclusion is again that lazy wavelets perform slightly worse than *k*-disk wavelets numerically, but there apparently is no significant difference between various values of *k*. Visually, there is no clear preference.

- ***Numerical stability:*** In the conversions to and from multiresolution form some numerical error is inevitable. While the numerical stability properties of orthogonal wavelet constructions are relatively well understood, stability of biorthogonal schemes like ours is less clear.

  Lacking theory to guide us, we ran the following experiment on the earth model. For each of the four types of wavelets we performed wavelet analysis followed by wavelet synthesis on a level $J = 6$ version of the model. For lazy wavelets, the relative error in the vertex positions was on the order of the machine precision. For 0-, 1-, and 2-disk wavelets, the relative errors were on the order of 0.00005, 0.001, and 0.002. When we reran the experiment using a level 3 version of the earth, the relative error for 2-disk wavelets was reduced to 1 in 10^6.

  Our conclusion is that wavelets with smaller supports are likely to be more stable numerically than those with larger ones, and that stability becomes increasingly important as the number of levels increases.

- ***Speed:*** Wavelets with larger support clearly take longer to add. There are potentially more new faces to introduce, and there are always a greater number of vertices whose hat function coefficients need to be updated. We ran a series of timing experiments and found that on average each type of wavelet could be added (the gather stage) at the following rate: lazy, 2700 coefficients per second; 0-disk, 2300; 1-disk, 1200; 2-disk, 600. The time for the evaluate stage was unchanged relative to wavelet size, which was expected.

Overall, we conclude that 0-disk wavelets combine good visual fidelity for a given number of coefficients and for a given number

of polygons, with good numerical stability and computation time. These findings, however, are preliminary, and require further confirmation.

5.1 Data encoding

As mentioned in the introduction, there is a small penalty for representing a mesh in multiresolution form. Since the wavelet coefficients are sorted in magnitude order for progressive transmission, we need to transmit with each coefficient the vertex identifier for the center of the wavelet. This information could be made implicit if the complete model was transmitted before any processing or display took place.

We use a simple encoding which represents the coefficient for a color or geometry wavelet with three floating point numbers, together with a word of information for the vertex identifier. This represents a 33% penalty for the benefit of progressive transmission. A suggestion for reducing this penalty is described below.

6 Discussion and future work

We have extended previous work on multiresolution analysis of meshes in two ways. First, we have shown how to perform multiresolution analysis of colored meshes by separately analyzing shape and color. Second, we have developed efficient algorithms and data structures that allow us to incrementally construct lower resolution approximations to colored meshes at interactive rates.

We have integrated these algorithms in a prototype mesh viewer that supports progressive transmission, dynamic display at a constant frame rate independent of machine performance and load, and the ability to interactively trade off the amount of detail in geometry and color. The separation of geometry and color also allows us to make efficient use of texture-mapping hardware.

In future work we intend to investigate:

- ***Multiresolution editing:*** In analogy to Finkelstein and Salesin's work on multiresolution curves [2] we plan to extend our multiresolution viewer to allow editing of meshes at different levels of detail.
- ***Other wavelets:*** We currently use piecewise linear wavelets to represent geometry and color. When modeling smooth objects or objects without sharp color transitions, use of smooth wavelets may result in better compression.
- ***Automatic tradeoff between color and geometry:*** If there is no texture-mapping hardware, adding wavelets for either color or geometry will increase the number of polygons that have to be rendered. When there is an upper bound on the number of polygons, for example during dynamic viewing, one has to choose between color detail and geometry detail. Currently the tradeoff is left to the user. Heuristics for automatically choosing a tradeoff that results in a visually close approximation would be useful.
- ***Comparison to progressive meshes:*** In simultaneous work Hoppe [3] has introduced the notion of *progressive meshes* to address the difficulties of storage, transmission, and display of complex meshes. The basic idea is to record the changes a mesh optimizer [4] makes as it simplifies a mesh. Since the original mesh can be recovered by running the record of changes in reverse, the progressive mesh representation is the simplest mesh together with the record of changes in reverse order. The relative advantages and disadvantages of such an approach need further study.
- ***Better encoding:*** The wavelet coefficients for a particular model typically span a large dynamic range, making floating point an obvious choice for encoding their values. Better use of bandwidth and storage could be made, however, by taking advantage of the wavelets being sorted in magnitude order. Fixed point numbers could be transmitted for each coefficient, with the scale information being transmitted only as it changes. This improvement could potentially eliminate the overhead incurred for progressive transmission.

Acknowledgments

This work was supported in part by the National Science Foundation under grants CCR-8957323, DMS-9103002, and DMS-9402734, by an Alfred P. Sloan Research Fellowship (BR-3495), an NSF Presidential Faculty Fellow award (CCR-9553199), an ONR Young Investigator award (N00014-95-1-0728), and industrial gifts from Interval, Microsoft, and Xerox. Head models courtesy of Cyberware.

References

[1] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 173–182. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[2] Adam Finkelstein and David Salesin. Multiresolution curves. *Computer Graphics (SIGGRAPH '94 Proceedings)*, 28(3):261–268, July 1994.

[3] H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[4] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In J.T. Kajiya, editor, *SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, pages 19–26. ACM SIGGRAPH, Addison Wesley, August 1993. held in Anaheim, California, 01-06 August 1993.

[5] J. Michael Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, Department of Computer Science and Engineering, University of Washington, September 1994. Available as ftp://cs.washington.edu/pub/graphics/LounsPhd.ps.Z.

[6] Michael Lounsbery, Tony DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *Submitted for publication*, 1994. Preliminary version available as Technical Report 93-10-05b, Department of Computer Science and Engineering, University of Washington, January, 1994. Also available as ftp://cs.washington.edu/pub/graphics/TR931005b.ps.Z.

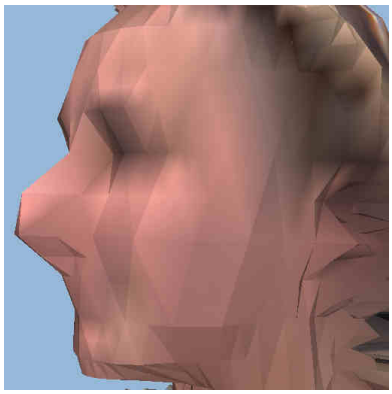
[7] Stephane Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[8] Yves Meyer. Ondelettes et fonctions splines. Technical report, *Séminaire EDP*, École Polytechnique, Paris, 1986.

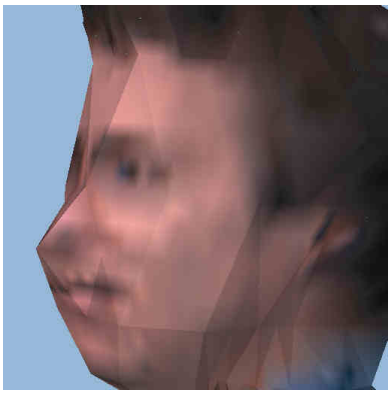
[9] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. In B. Falcidieno and T.L. Kunii, editors, *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, June-July 1993.

[10] Peter Schröder and Wim Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 161–172. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

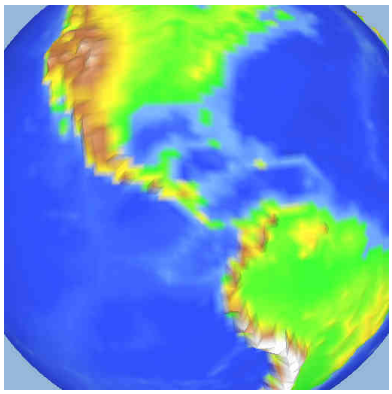
[11] Eric Stollnitz, Tony DeRose, and David Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan–Kaufmann, 1996.



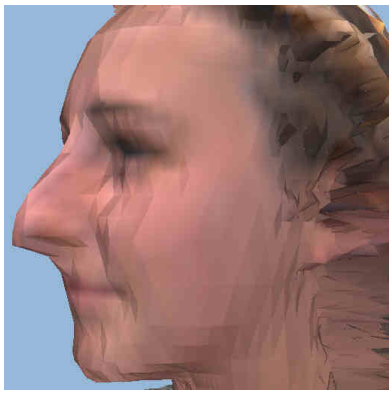
a) 3 seconds...



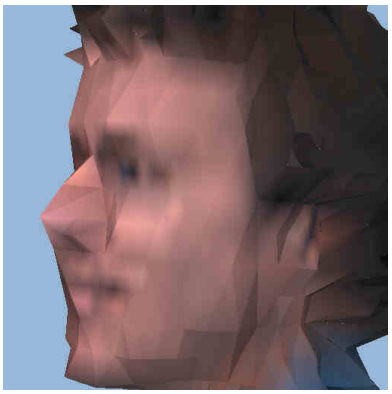
e) high color detail



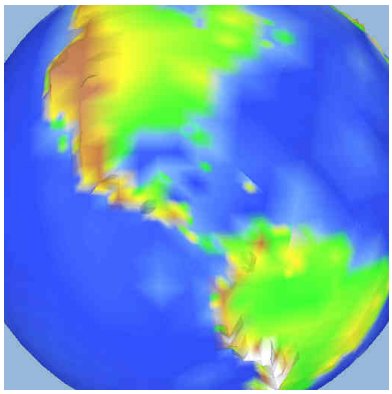
i) full resolution



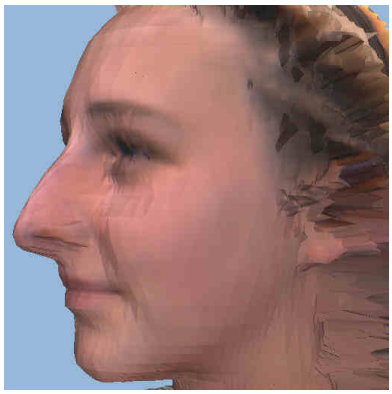
b) 17 seconds...



f) equal color and geometry detail

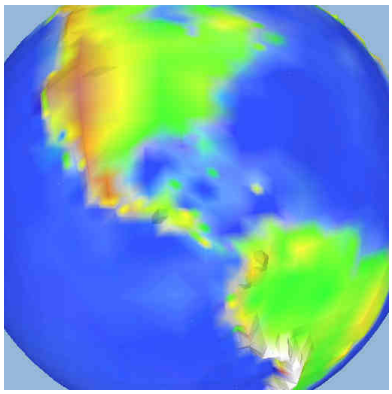


j) lazy wavelets

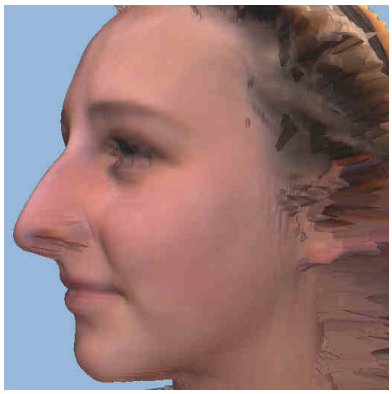


c) 59 seconds...

g) high geometry detail

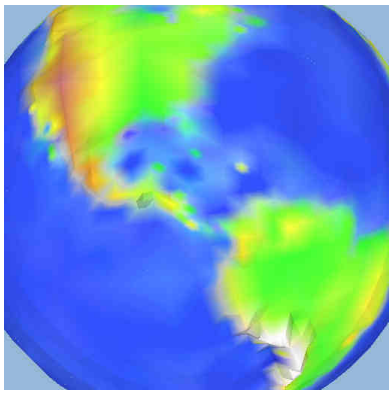


k) 0-disk wavelets



d) full resolution

h) texture mapping



l) 2-disk wavelets

Color Plate 1