Progressive Meshes

Hugues Hoppe
Microsoft Research

ABSTRACT

Highly detailed geometric models are rapidly becoming commonplace in computer graphics. These models, often represented as complex triangle meshes, challenge rendering performance, transmission bandwidth, and storage capacities. This paper introduces the *progressive mesh* (PM) representation, a new scheme for storing and transmitting arbitrary triangle meshes. This efficient, lossless, continuous-resolution representation addresses several practical problems in graphics: smooth geomorphing of level-of-detail approximations, progressive transmission, mesh compression, and selective refinement.

In addition, we present a new mesh simplification procedure for constructing a PM representation from an arbitrary mesh. The goal of this optimization procedure is to preserve not just the geometry of the original mesh, but more importantly its overall appearance as defined by its discrete and scalar appearance attributes such as material identifiers, color values, normals, and texture coordinates. We demonstrate construction of the PM representation and its applications using several practical models.

CR Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - surfaces and object representations.

Additional Keywords: mesh simplification, level of detail, shape interpolation, progressive transmission, geometry compression.

1 INTRODUCTION

Highly detailed geometric models are necessary to satisfy a growing expectation for realism in computer graphics. Within traditional modeling systems, detailed models are created by applying versatile modeling operations (such as extrusion, constructive solid geometry, and freeform deformations) to a vast array of geometric primitives. For efficient display, these models must usually be tessellated into polygonal approximations—meshes. Detailed meshes are also obtained by scanning physical objects using range scanning systems [5]. In either case, the resulting complex meshes are expensive to store, transmit, and render, thus motivating a number of practical problems:

- *Mesh simplification*: The meshes created by modeling and scanning systems are seldom optimized for rendering efficiency, and can frequently be replaced by nearly indistinguishable approximations with far fewer faces. At present, this process often requires significant user intervention. Mesh simplification tools can hope to automate this painstaking task, and permit the porting of a single model to platforms of varying performance.
- *Level-of-detail (LOD) approximation*: To further improve rendering performance, it is common to define several versions of a model at various levels of detail [3, 8]. A detailed mesh is used when the object is close to the viewer, and coarser approximations are substituted as the object recedes. Since instantaneous switching between LOD meshes may lead to perceptible "popping", one would like to construct smooth visual transitions, *geomorphs*, between meshes at different resolutions.
- *Progressive transmission*: When a mesh is transmitted over a communication line, one would like to show progressively better approximations to the model as data is incrementally received. One approach is to transmit successive LOD approximations, but this requires additional transmission time.
- *Mesh compression*: The problem of minimizing the storage space for a model can be addressed in two orthogonal ways. One is to use mesh simplification to reduce the number of faces. The other is mesh compression: minimizing the space taken to store a particular mesh.
- *Selective refinement*: Each mesh in a LOD representation captures the model at a uniform (view-independent) level of detail. Sometimes it is desirable to adapt the level of refinement in selected regions. For instance, as a user flies over a terrain, the terrain mesh need be fully detailed only near the viewer, and only within the field of view.

In addressing these problems, this paper makes two major contributions. First, it introduces the *progressive mesh* (PM) representation. In PM form, an arbitrary mesh $\hat{M}$ is stored as a much coarser mesh M^0 together with a sequence of n detail records that indicate how to incrementally refine M^0 exactly back into the original mesh $\hat{M} = M^n$. Each of these records stores the information associated with a *vertex split*, an elementary mesh transformation that adds an additional vertex to the mesh. The PM representation of $\hat{M}$ thus defines a continuous sequence of meshes $M^0, M^1, \ldots, M^n$ of increasing accuracy, from which LOD approximations of any desired complexity can be efficiently retrieved. Moreover, geomorphs can be efficiently constructed between any two such meshes. In addition, we show that the PM representation naturally supports progressive transmission, offers a concise encoding of $\hat{M}$ itself, and permits selective refinement. In short, progressive meshes offer an efficient, lossless, continuous-resolution representation.

The other contribution of this paper is a new simplification procedure for constructing a PM representation from a given mesh $\hat{M}$. Unlike previous simplification methods, our procedure seeks to preserve not just the geometry of the mesh surface, but more importantly its overall appearance, as defined by the discrete and scalar attributes associated with its surface.

Email: hhoppe@microsoft.com
Web: http://www.research.microsoft.com/research/graphics/hoppe/

2 MESHES IN COMPUTER GRAPHICS

Models in computer graphics are often represented using triangle meshes.[1] Geometrically, a triangle mesh is a piecewise linear surface consisting of triangular faces pasted together along their edges. As described in [9], the mesh geometry can be denoted by a tuple (K, V), where K is a *simplicial complex* specifying the connectivity of the mesh simplices (the adjacency of the vertices, edges, and faces), and $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ is the set of vertex positions defining the shape of the mesh in $\mathbf{R}^3$. More precisely (cf. [9]), we construct a parametric domain $|K| \subset \mathbf{R}^m$ by identifying each vertex of K with a canonical basis vector of $\mathbf{R}^m$, and define the mesh as the image $\phi_V(|K|)$ where $\phi_V : \mathbf{R}^m \to \mathbf{R}^3$ is a linear map.

Often, surface appearance attributes other than geometry are also associated with the mesh. These attributes can be categorized into two types: *discrete* attributes and *scalar* attributes.

Discrete attributes are usually associated with faces of the mesh. A common discrete attribute, the *material identifier*, determines the shader function used in rendering a face of the mesh [18]. For instance, a trivial shader function may involve simple look-up within a specified texture map.

Many scalar attributes are often associated with a mesh, including diffuse color (r, g, b), normal (n_x, n_y, n_z), and texture coordinates (u, v). More generally, these attributes specify the local parameters of shader functions defined on the mesh faces. In simple cases, these scalar attributes are associated with vertices of the mesh. However, to represent discontinuities in the scalar fields, and because adjacent faces may have different shading functions, it is common to associate scalar attributes not with vertices, but with corners of the mesh [1]. A *corner* is defined as a (vertex,face) tuple. Scalar attributes at a corner (v, f) specify the shading parameters for face f at vertex v. For example, along a *crease* (a curve on the surface across which the normal field is not continuous), each vertex has two distinct normals, one associated with the corners on each side of the crease.

We express a mesh as a tuple $M = (K, V, D, S)$ where V specifies its geometry, D is the set of discrete attributes d_f associated with the faces $f = \{j, k, l\} \in K$, and S is the set of scalar attributes $s_{(v,f)}$ associated with the corners (v, f) of K.

The attributes D and S give rise to discontinuities in the visual appearance of the mesh. An edge $\{v_j, v_k\}$ of the mesh is said to be *sharp* if either (1) it is a boundary edge, or (2) its two adjacent faces f_l and f_r have different discrete attributes (i.e. $d_{f_l} \neq d_{f_r}$), or (3) its adjacent corners have different scalar attributes (i.e. $s_{(v_j,f_l)} \neq s_{(v_j,f_r)}$ or $s_{(v_k,f_l)} \neq s_{(v_k,f_r)}$). Together, the set of sharp edges define a set of *discontinuity curves* over the mesh (e.g. the yellow curves in Figure 12).

3 PROGRESSIVE MESH REPRESENTATION

3.1 Overview

Hoppe et al. [9] describe a method, *mesh optimization*, that can be used to approximate an initial mesh $\hat{M}$ by a much simpler one. Their optimization algorithm, reviewed in Section 4.1, traverses the space of possible meshes by successively applying a set of 3 mesh transformations: edge collapse, edge split, and edge swap.

We have discovered that in fact a single one of those transformations, *edge collapse*, is sufficient for effectively simplifying meshes. As shown in Figure 1, an edge collapse transformation $ecol(\{v_s, v_t\})$ unifies 2 adjacent vertices v_s and v_t into a single vertex v_s. The vertex v_t and the two adjacent faces $\{v_s, v_t, v_l\}$ and $\{v_t, v_s, v_r\}$ vanish in the process. A position $\mathbf{v}_s$ is specified for the new unified vertex.

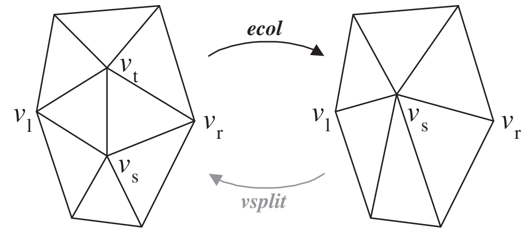


Figure 1: Illustration of the edge collapse transformation.

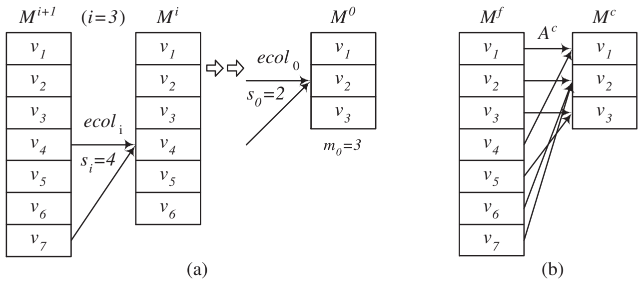


Figure 2: (a) Sequence of edge collapses; (b) Resulting vertex correspondence.

Thus, an initial mesh $\hat{M} = M^n$ can be simplified into a coarser mesh M^0 by applying a sequence of n successive edge collapse transformations:

$$(\hat{M} = M^n) \xrightarrow{ecol_{n-1}} \ldots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0 .$$

The particular sequence of edge collapse transformations must be chosen carefully, since it determines the quality of the approximating meshes $M^i, i < n$. A scheme for choosing these edge collapses is presented in Section 4.

Let m_0 be the number of vertices in M^0, and let us label the vertices of mesh M^i as $V^i = \{v_1, \ldots, v_{m_0+i}\}$, so that edge $\{v_{s_i}, v_{m_0+i+1}\}$ is collapsed by $ecol_i$ as shown in Figure 2a. As vertices may have different positions in the different meshes, we denote the position of v_j in M^i as $\mathbf{v}_j^i$.

A key observation is that an edge collapse transformation is invertible. Let us call that inverse transformation a *vertex split*, shown as *vsplit* in Figure 1. A vertex split transformation $vsplit(s, l, r, t, A)$ adds near vertex v_s a new vertex v_t and two new faces $\{v_s, v_t, v_l\}$ and $\{v_t, v_s, v_r\}$. (If the edge $\{v_s, v_t\}$ is a boundary edge, we let $v_r = 0$ and only one face is added.) The transformation also updates the attributes of the mesh in the neighborhood of the transformation. This attribute information, denoted by A, includes the positions $\mathbf{v}_s$ and $\mathbf{v}_t$ of the two affected vertices, the discrete attributes $d_{\{v_s,v_t,v_l\}}$ and $d_{\{v_t,v_s,v_r\}}$ of the two new faces, and the scalar attributes of the affected corners ($s_{(v_s,\cdot)}$, $s_{(v_t,\cdot)}$, $s_{(v_l,\{v_s,v_t,v_l\})}$, and $s_{(v_r,\{v_t,v_s,v_r\})}$).

Because edge collapse transformations are invertible, we can therefore represent an arbitrary triangle mesh $\hat{M}$ as a simple mesh M^0 together with a sequence of n *vsplit* records:

$$M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} \ldots \xrightarrow{vsplit_{n-1}} (M^n = \hat{M})$$

where each record is parametrized as $vsplit_i(s_i, l_i, r_i, A_i)$. We call $(M^0, \{vsplit_0, \ldots, vsplit_{n-1}\})$ a *progressive mesh* (PM) representation of $\hat{M}$.

As an example, the mesh $\hat{M}$ of Figure 5d (13,546 faces) was simplified down to the coarse mesh M^0 of Figure 5a (150 faces) using

[1] We assume in this paper that more general meshes, such as those containing n-sided faces and faces with holes, are first converted into triangle meshes by triangulation. The PM representation could be generalized to handle the more general meshes directly, at the expense of more complex data structures.

6,698 edge collapse transformations. Thus its PM representation consists of M^0 together with a sequence of $n=6698$ *vsplit* records. From this PM representation, one can extract approximating meshes with any desired number of faces (actually, within ± 1) by applying to M^0 a prefix of the *vsplit* sequence. For example, Figure 5 shows approximating meshes with 150, 500, and 1000 faces.

3.2 Geomorphs

A nice property of the vertex split transformation (and its inverse, edge collapse) is that a smooth visual transition (a *geomorph*) can be created between the two meshes M^i and M^{i+1} in $M^i \xrightarrow{vsplit_i} M^{i+1}$. For the moment let us assume that the meshes contain no attributes other than vertex positions. With this assumption the vertex split record is encoded as $vsplit_i(s_i, l_i, r_i, A_i = (\mathbf{v}_{s_i}^{i+1}, \mathbf{v}_{m_0+i+1}^{i+1}))$. We construct a geomorph $M^G(\alpha)$ with blend parameter $0 \le \alpha \le 1$ such that $M^G(0)$ looks like M^i and $M^G(1)$ looks like M^{i+1}—in fact $M^G(1)=M^{i+1}$—by defining a mesh

$$M^G(\alpha) = (K^{i+1}, V^G(\alpha))$$

whose connectivity is that of M^{i+1} and whose vertex positions linearly interpolate from $v_{s_i} \in M^i$ to the split vertices $v_{s_i}, v_{m_0+i+1} \in M^{i+1}$:

$$\mathbf{v}_j^G(\alpha) = \begin{cases} (\alpha)\mathbf{v}_j^{i+1} + (1-\alpha)\mathbf{v}_{s_i}^i & , j \in \{s_i, m_0+i+1\} \\ \mathbf{v}_j^{i+1} = \mathbf{v}_j^i & , j \notin \{s_i, m_0+i+1\} \end{cases}$$

Using such geomorphs, an application can smoothly transition from a mesh M^i to meshes M^{i+1} or M^{i-1} without any visible "snapping" of the meshes.

Moreover, since individual *ecol* transformations can be transitioned smoothly, so can the composition of any sequence of them. Geomorphs can therefore be constructed between *any* two meshes of a PM representation. Indeed, given a finer mesh M^f and a coarser mesh M^c, $0 \le c < f \le n$, there exists a natural correspondence between their vertices: each vertex of M^f is related to a unique ancestor vertex of M^c by a surjective map A^c obtained by composing a sequence of *ecol* transformations (Figure 2b). More precisely, each vertex v_j of M^f corresponds with the vertex $v_{A^c(j)}$ in M^c where

$$A^c(j) = \begin{cases} j & , j \le m_0 + c \\ A^c(s_{j-m_0-1}) & , j > m_0 + c \end{cases} .$$

(In practice, this ancestor information A^c is gathered in a forward fashion as the mesh is refined.) This correspondence allows us to define a geomorph $M^G(\alpha)$ such that $M^G(0)$ looks like M^c and $M^G(1)$ equals M^f. We simply define $M^G(\alpha) = (K^f, V^G(\alpha))$ to have the connectivity of M^f and the vertex positions

$$\mathbf{v}_j^G(\alpha) = (\alpha)\mathbf{v}_j^f + (1-\alpha)\mathbf{v}_{A^c(j)}^c .$$

So far we have outlined the construction of geomorphs between PM meshes containing only position attributes. We can in fact construct geomorphs for meshes containing both discrete and scalar attributes.

Discrete attributes by their nature cannot be smoothly interpolated. Fortunately, these discrete attributes are associated with faces of the mesh, and the "geometric" geomorphs described above smoothly introduce faces. In particular, observe that the faces of M^c are a proper subset of the faces of M^f, and that those faces of M^f missing from M^c are invisible in $M^G(0)$ because they have been collapsed to degenerate (zero area) triangles. Other geomorphing schemes [10, 11, 17] define well-behaved (invertible) parametrizations between meshes at different levels of detail, but these do not permit the construction of geomorphs between meshes with different discrete attributes.

Scalar attributes defined on corners can be smoothly interpolated much like the vertex positions. There is a slight complication in that a corner (v,f) in a mesh M is not naturally associated with any "ancestor corner" in a coarser mesh M^c if f is not a face of M^c. We can still attempt to infer what attribute value (v,f) would have in M^c as follows. We examine the mesh M^{i+1} in which f is first introduced, locate a neighboring corner (v,f') in M^{i+1} whose attribute value is the same, and recursively backtrack from it to a corner in M^c. If there is no neighboring corner in M^{i+1} with an identical attribute value, then the corner (v,f) has no equivalent in M^c and we therefore keep its attribute value constant through the geomorph.

The interpolating function on the scalar attributes need not be linear; for instance, normals are best interpolated over the unit sphere, and colors may be interpolated in a color space other than RGB.

Figure 6 demonstrates a geomorph between two meshes M^{175} (500 faces) and M^{425} (1000 faces) retrieved from the PM representation of the mesh in Figure 5d.

3.3 Progressive transmission

Progressive meshes are a natural representation for progressive transmission. The compact mesh M^0 is transmitted first (using a conventional uni-resolution format), followed by the stream of $vsplit_i$ records. The receiving process incrementally rebuilds $\hat{M}$ as the records arrive, and animates the changing mesh. The changes to the mesh can be geomorphed to avoid visual discontinuities. The original mesh $\hat{M}$ is recovered exactly after all n records are received, since PM is a lossless representation.

The computation of the receiving process should be balanced between the reconstruction of $\hat{M}$ and interactive display. With a slow communication line, a simple strategy is to display the current mesh whenever the input buffer is found to be empty. With a fast communication line, we find that a good strategy is to display meshes whose complexities increase exponentially. (Similar issues arise in the display of images transmitted using progressive JPEG.)

3.4 Mesh compression

Even though the PM representation encodes both $\hat{M}$ and a continuous family of approximations, it is surprisingly space-efficient, for two reasons. First, the locations of the vertex split transformations can be encoded concisely. Instead of storing all three vertex indices (s_i, l_i, r_i) of $vsplit_i$, one need only store s_i and approximately 5 bits to select the remaining two vertices among those adjacent to v_{s_i}.[2] Second, because a vertex split has local effect, one can expect significant coherence in mesh attributes through each transformation. For instance, when vertex $v_{s_i}^i$ is split into $v_{s_i}^{i+1}$ and $v_{m_0+i+1}^{i+1}$, we can predict the positions $\mathbf{v}_{s_i}^{i+1}$ and $\mathbf{v}_{m_0+i+1}^{i+1}$ from $\mathbf{v}_{s_i}^i$, and use delta-encoding to reduce storage. Scalar attributes of corners in M^{i+1} can similarly be predicted from those in M^i. Finally, the material identifiers $d_{\{v_s,v_t,v_l\}}$ and $d_{\{v_t,v_s,v_r\}}$ of the new faces in mesh M^{i+1} can often be predicted from those of adjacent faces in M^i using only a few control bits.

As a result, the size of a carefully designed PM representation should be competitive with that obtained from methods for compressing uni-resolution meshes. Our current prototype implementation was not designed with this goal in mind. However, we analyze the compression of the connectivity K, and report results on the compression of the geometry V. In the following analysis, we assume for simplicity that $m_0 = 0$ since typically $m_0 \ll n$.

A common representation for the mesh connectivity K is to list the three vertex indices for each face. Since the number of vertices is n and the number of faces approximately $2n$, such a list requires $6\lceil \log_2(n) \rceil n$ bits of storage. Using a buffer of 2 vertices, *generalized triangle strip* representations reduce this number to about

[2] On average, v_{s_i} has 6 neighbors, and the number of permutations $P_2^6=30$ can be encoded in $\lceil \log_2(P_2^6) \rceil = 5$ bits.

$(\lceil \log_2(n) \rceil + 2k)n$ bits, where vertices are back-referenced once on average and $k \simeq 2$ bits capture the vertex replacement codes [6]. By increasing the vertex buffer size to 16, Deering's *generalized triangle mesh* representation [6] further reduces storage to about $(\frac{1}{8}\lceil \log_2(n) \rceil + 8)n$ bits. Turan [16] shows that planar graphs (and hence the connectivity of closed genus 0 meshes) can be encoded in $12n$ bits. Recent work by Taubin and Rossignac [15] addresses more general meshes. With the PM representation, each $vsplit_i$ requires specification of s_i and its two neighbors, for a total storage of about $(\lceil \log_2(n) \rceil + 5)n$ bits. Although not as concise as [6, 15], this is comparable to generalized triangle strips.

A traditional representation of the mesh geometry V requires storage of $3n$ coordinates, or $96n$ bits with IEEE single-precision floating point. Like Deering [6], we assume that these coordinates can be quantized to 16-bit fixed precision values without significant loss of visual quality, thus reducing storage to $48n$ bits. Deering is able to further compress this storage by delta-encoding the quantized coordinates and Huffman compressing the variable-length deltas. For 16-bit quantization, he reports storage of $35.8n$ bits, which includes both the deltas and the Huffman codes. Using a similar approach with the PM representation, we encode V in $31n$ to $50n$ bits as shown in Table 1. To obtain these results, we exploit a property of our optimization algorithm (Section 4.3): when considering the collapse of an edge $\{v_s, v_t\}$, it considers three starting points for the resulting vertex position $\mathbf{v}_n$: $\{\mathbf{v}_s, \mathbf{v}_t, \frac{\mathbf{v}_s + \mathbf{v}_t}{2}\}$. Depending on the starting point chosen, we delta-encode either $\{\mathbf{v}_s - \mathbf{v}_n, \mathbf{v}_t - \mathbf{v}_n\}$ or $\{\frac{\mathbf{v}_s + \mathbf{v}_t}{2} - \mathbf{v}_n, \frac{\mathbf{v}_t - \mathbf{v}_s}{2}\}$, and use separate Huffman tables for all four quantities.

To further improve compression, we could alter the construction algorithm to forego optimization and let $\mathbf{v}_n \in \{\mathbf{v}_s, \mathbf{v}_t, \frac{\mathbf{v}_s + \mathbf{v}_t}{2}\}$. This would degrade the accuracy of the approximating meshes somewhat, but allows encoding of V in $30n$ to $37n$ bits in our examples. Arithmetic coding [19] of delta lengths does not improve results significantly, reflecting the fact that the Huffman trees are well balanced. Further compression improvements may be achievable by adapting both the quantization level and the delta length models as functions of the *vsplit* record index i, since the magnitude of successive changes tends to decrease.

3.5 Selective refinement

The PM representation also supports selective refinement, whereby detail is added to the model only in desired areas. Let the application supply a callback function REFINE(v) that returns a Boolean value indicating whether the neighborhood of the mesh about v should be further refined. An initial mesh M^c is selectively refined by iterating through the list $\{vsplit_c, \ldots, vsplit_{n-1}\}$ as before, but only performing $vsplit_i(s_i, l_i, r_i, A_i)$ if

(1) all three vertices $\{v_{s_i}, v_{l_i}, v_{r_i}\}$ are present in the mesh, and

(2) REFINE(v_{s_i}) evaluates to TRUE.

(A vertex v_j is absent from the mesh if the prior vertex split that would have introduced it, $vsplit_{j-m_0-1}$, was not performed due to the above conditions.)

As an example, to obtain selective refinement of the model within a view frustum, REFINE(v) is defined to be TRUE if either v or any of its neighbors lies within the frustum. As seen in Figure 7a, condition (1) described above is suboptimal. The problem is that a vertex v_{s_i} within the frustum may fail to be split because its expected neighbor v_{l_i} lies just outside the frustum and was not previously created. The problem is remedied by using a less stringent version of condition (1). Let us define the *closest living ancestor* of a vertex v_j to be the vertex with index

$$A'(j) = \begin{cases} j & \text{, if } v_j \text{ exists in the mesh} \\ A'(s_{j-m_0-1}) & \text{, otherwise} \end{cases}$$

The new condition becomes:

(1') v_{s_i} is present in the mesh (i.e. $A'(s_i) = s_i$) and the vertices $v_{A'(l_i)}$ and $v_{A'(r_i)}$ are both adjacent to v_{s_i}.

As when constructing the geomorphs, the ancestor information A' is carried efficiently as the *vsplit* records are parsed. If conditions (1') and (2) are satisfied, $vsplit(s_i, A'(l_i), A'(r_i), A_i)$ is applied to the mesh. A mesh selectively refined with this new strategy is shown in Figure 7b. This same strategy was also used for Figure 10. Note that it is still possible to create geomorphs between M^c and selectively refined meshes thus created.

An interesting application of selective refinement is the transmission of view-dependent models over low-bandwidth communication lines. As the receiver's view changes over time, the sending process need only transmit those *vsplit* records for which REFINE evaluates to TRUE, and of those only the ones not previously transmitted.

4 PROGRESSIVE MESH CONSTRUCTION

The PM representation of an arbitrary mesh $\hat{M}$ requires a sequence of edge collapses transforming $\hat{M} = M^n$ into a base mesh M^0. The quality of the intermediate approximations $M^i, i < n$ depends largely on the algorithm for selecting which edges to collapse and what attributes to assign to the affected neighborhoods, for instance the positions $\mathbf{v}^i_{s_i}$.

There are many possible PM construction algorithms with varying trade-offs of speed and accuracy. At one extreme, a crude and fast scheme for selecting edge collapses is to choose them completely at random. (Some local conditions must be satisfied for an edge collapse to be legal, i.e. manifold preserving [9].) More sophisticated schemes can use heuristics to improve the edge selection strategy, for example the "distance to plane" metric of Schroeder et al. [14]. At the other extreme, one can attempt to find approximating meshes that are optimal with respect to some appearance metric, for instance the E_{dist} geometric metric of Hoppe et al. [9].

Since PM construction is a preprocess that can be performed offline, we chose to design a simplification procedure that invests some time in the selection of edge collapses. Our procedure is similar to the mesh optimization method introduced by Hoppe et al. [9], which is outlined briefly in Section 4.1. Section 4.2 presents an overview of our procedure, and Sections 4.3–4.6 present the details of our optimization scheme for preserving both the shape of the mesh and the scalar and discrete attributes which define its appearance.

4.1 Background: mesh optimization

The goal of mesh optimization [9] is to find a mesh $M = (K, V)$ that both accurately fits a set X of points $\mathbf{x}_i \in \mathbf{R}^3$ and has a small number of vertices. This problem is cast as minimization of an energy function

$$E(M) = E_{dist}(M) + E_{rep}(M) + E_{spring}(M) .$$

The first two terms correspond to the two goals of accuracy and conciseness: the *distance energy* term

$$E_{dist}(M) = \sum_i d^2(\mathbf{x}_i, \phi_V(|K|))$$

measures the total squared distance of the points from the mesh, and the *representation energy* term $E_{rep}(M) = c_{rep} m$ penalizes the number m of vertices in M. The third term, the *spring energy* $E_{spring}(M)$ is introduced to regularize the optimization problem. It corresponds to placing on each edge of the mesh a spring of rest length zero and tension κ:

$$E_{spring}(M) = \sum_{\{j,k\} \in K} \kappa \|\mathbf{v}_j - \mathbf{v}_k\|^2 .$$

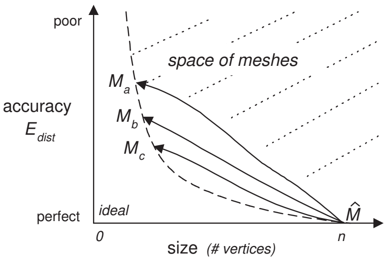


Figure 3: Illustration of the paths taken by mesh optimization using three different settings of c_{rep}.

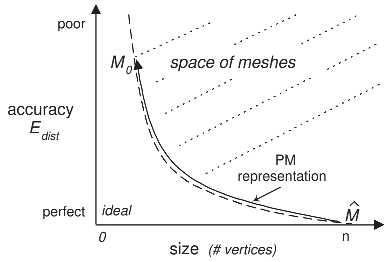


Figure 4: Illustration of the path taken by the new mesh simplification procedure in a graph plotting accuracy vs. mesh size.

The energy function $E(M)$ is minimized using a nested optimization method:

- *Outer loop*: The algorithm optimizes over K, the connectivity of the mesh, by randomly attempting a set of three possible mesh transformations: edge collapse, edge split, and edge swap. This set of transformations is complete, in the sense that any simplicial complex K of the same topological type as $\hat{K}$ can be reached through a sequence of these transformations. For each candidate mesh transformation, $K \rightarrow K'$, the continuous optimization described below computes $E_{K'}$, the minimum of E subject to the new connectivity K'. If $\Delta E = E_{K'} - E_K$ is found to be negative, the mesh transformation is applied (akin to a zero-temperature simulated annealing method).
- *Inner loop*: For each candidate mesh transformation, the algorithm computes $E_{K'} = \min_V E_{dist}(V) + E_{spring}(V)$ by optimizing over the vertex positions V. For the sake of efficiency, the algorithm in fact optimizes only one vertex position $\mathbf{v}_s$, and considers only the subset of points X that project onto the neighborhood affected by $K \rightarrow K'$. To avoid surface self-intersections, the edge collapse is disallowed if the maximum dihedral angle of edges in the resulting neighborhood exceeds some threshold.

Hoppe et al. [9] find that the regularizing spring energy term $E_{spring}(M)$ is most important in the early stages of the optimization, and achieve best results by repeatedly invoking the nested optimization method described above with a schedule of decreasing spring constants κ.

Mesh optimization is demonstrated to be an effective tool for mesh simplification. Given an initial mesh $\hat{M}$ to approximate, a dense set of points X is sampled both at the vertices of $\hat{M}$ and randomly over its faces. The optimization algorithm is then invoked with $\hat{M}$ as the starting mesh. Varying the setting of the representation constant c_{rep} results in optimized meshes with different trade-offs of accuracy and size. The paths taken by these optimizations are shown illustratively in Figure 3.

4.2 Overview of the simplification algorithm

As in mesh optimization [9], we also define an explicit energy metric $E(M)$ to measure the accuracy of simplified meshes $M = (K, V, D, S)$ with respect to the original $\hat{M}$, and we also modify the mesh M starting from $\hat{M}$ while minimizing $E(M)$.

Our energy metric has the following form:

$$E(M) = E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M) \ .$$

The first two terms, $E_{dist}(M)$ and $E_{spring}(M)$ are identical to those in [9]. The next two terms of $E(M)$ are added to preserve attributes associated with M: $E_{scalar}(M)$ measures the accuracy of its scalar attributes (Section 4.4), and $E_{disc}(M)$ measures the geometric accuracy of its discontinuity curves (Section 4.5). (To achieve scale invariance of the terms, the mesh is uniformly scaled to fit in a unit cube.)

Our scheme for optimizing over the connectivity K of the mesh is rather different from [9]. We have discovered that a mesh can be effectively simplified using edge collapse transformations alone. The edge swap and edge split transformations, useful in the context of surface reconstruction (which motivated [9]), are not essential for simplification. Although in principle our simplification algorithm can no longer traverse the entire space of meshes considered by mesh optimization, we find that the meshes generated by our algorithm are just as good. In fact, because of the priority queue approach described below, our meshes are usually better. Moreover, considering only edge collapses simplifies the implementation, improves performance, and most importantly, gives rise to the PM representation (Section 3).

Rather than randomly attempting mesh transformations as in [9], we place all (legal) candidate edge collapse transformations into a priority queue, where the priority of each transformation is its estimated energy cost ΔE. In each iteration, we perform the transformation at the front of the priority queue (with lowest ΔE), and recompute the priorities of edges in the neighborhood of this transformation. As a consequence, we eliminate the need for the awkward parameter c_{rep} as well as the energy term $E_{rep}(M)$. Instead, we can explicitly specify the number of faces desired in an optimized mesh. Also, a single run of the optimization can generate several such meshes. Indeed, it generates a continuous-resolution family of meshes, namely the PM representation of $\hat{M}$ (Figure 4).

For each edge collapse $K \rightarrow K'$, we compute its cost $\Delta E = E_{K'} - E_K$ by solving a continuous optimization

$$E_{K'} = \min_{V,S} E_{dist}(V) + E_{spring}(V) + E_{scalar}(V, S) + E_{disc}(V)$$

over both the vertex positions V and the scalar attributes S of the mesh with connectivity K'. This minimization is discussed in the next three sections.

4.3 Preserving surface geometry ($E_{dist} + E_{spring}$)

As in [9], we "record" the geometry of the original mesh $\hat{M}$ by sampling from it a set of points X. At a minimum, we sample a point at each vertex of $\hat{M}$. If requested by the user, additional points are sampled randomly over the surface of $\hat{M}$. The energy terms $E_{dist}(M)$ and $E_{spring}(M)$ are defined as in Section 4.1.

For a mesh of fixed connectivity, our method for optimizing the vertex positions to minimize $E_{dist}(V) + E_{spring}(V)$ closely follows that of [9]. Evaluating $E_{dist}(V)$ involves computing the distance of each point $\mathbf{x}_i$ to the mesh. Each of these distances is itself a minimization problem

$$d^2(\mathbf{x}_i, \phi_V(|K|)) = \min_{\mathbf{b}_i \in |K|} \|\mathbf{x}_i - \phi_V(\mathbf{b}_i)\|^2 \qquad (1)$$

where the unknown $\mathbf{b}_i$ is the parametrization of the projection of $\mathbf{x}_i$ on the mesh. The nonlinear minimization of $E_{dist}(V) + E_{spring}(V)$ is performed using an iterative procedure alternating between two steps:

1. For fixed vertex positions V, compute the optimal parametrizations $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_{|X|}\}$ by projecting the points X onto the mesh.
2. For fixed parametrizations B, compute the optimal vertex positions V by solving a sparse linear least-squares problem.

As in [9], when considering $ecol(\{v_s, v_t\})$, we optimize only one vertex position, $\mathbf{v}_s^i$. We perform three different optimizations with different starting points, $\mathbf{v}_s^i = (1-\alpha)\mathbf{v}_s^{i+1} + (\alpha)\mathbf{v}_t^{i+1}$ for $\alpha = \{0, \frac{1}{2}, 1\}$, and accept the best one.

Instead of defining a global spring constant κ for E_{spring} as in [9], we adapt κ each time an edge collapse transformation is considered. Intuitively, the spring energy is most important when few points project onto a neighborhood of faces, since in this case finding the vertex positions minimizing $E_{dist}(V)$ may be an under-constrained problem. Thus, for each edge collapse transformation considered, we set κ as a function of the ratio of the number of points to the number of faces in the neighborhood.[3] With this adaptive scheme, the influence of $E_{spring}(M)$ decreases gradually and adaptively as the mesh is simplified, and we no longer require the expensive schedule of decreasing spring constants.

4.4 Preserving scalar attributes (E_{scalar})

As described in Section 2, we represent piecewise continuous scalar fields by defining scalar attributes S at the mesh corners. We now present our scheme for preserving these scalar fields through the simplification process. For exposition, we find it easier to first present the case of continuous scalar fields, in which the corner attributes at a vertex are identical. The generalization to piecewise continuous fields is discussed shortly.

Optimizing scalar attributes at vertices Let the original mesh $\hat{M}$ have at each vertex v_j not only a position $\mathbf{v}_j \in \mathbf{R}^3$ but also a scalar attribute $\underline{\mathbf{v}}_j \in \mathbf{R}^d$. To capture scalar attributes, we sample at each point $\mathbf{x}_i \in X$ the attribute value $\underline{\mathbf{x}}_i \in \mathbf{R}^d$. We would then like to generalize the distance metric E_{dist} to also measure the deviation of the sampled attribute values $\underline{X}$ from those of M.

One natural way to achieve this is to redefine the distance metric to measure distance in $\mathbf{R}^{3+d}$:

$$d^2((\mathbf{x}_i \;\; \underline{\mathbf{x}}_i), M(K, V, \underline{V})) = \min_{\mathbf{b}_i \in |K|} \|(\mathbf{x}_i \;\; \underline{\mathbf{x}}_i) - (\phi_V(\mathbf{b}_i) \;\; \phi_{\underline{V}}(\mathbf{b}_i))\|^2 \ .$$

This new distance functional could be minimized using the iterative approach of Section 4.3. However, it would be expensive since finding the optimal parametrization $\mathbf{b}_i$ of each point $\mathbf{x}_i$ would require projection in $\mathbf{R}^{3+d}$, and would be non-intuitive since these parametrizations would not be geometrically based.

Instead we opted to determine the parametrizations $\mathbf{b}_i$ using only geometry with equation (1), and to introduce a separate energy term E_{scalar} to measure attribute deviation based on these parametrizations:

$$E_{scalar}(\underline{V}) = (c_{scalar})^2 \sum_i \|\underline{\mathbf{x}}_i - \phi_{\underline{V}}(\mathbf{b}_i)\|^2$$

where the constant c_{scalar} assigns a relative weight between the scalar attribute errors (E_{scalar}) and the geometric errors (E_{dist}).

Thus, to minimize $E(V, \underline{V}) = E_{dist}(V) + E_{spring}(V) + E_{scalar}(\underline{V})$, our algorithm first finds the vertex position $\mathbf{v}_s$ minimizing $E_{dist}(V) + E_{spring}(V)$ by alternately projecting the points onto the mesh (obtaining the parametrizations $\mathbf{b}_i$) and solving a linear least-squares problem (Section 4.1). Then, using those same parametrizations $\mathbf{b}_i$, it finds the vertex attribute $\underline{\mathbf{v}}_s$ minimizing E_{scalar} by solving a single linear least-squares problem. Hence introducing E_{scalar} into the optimization causes negligible performance overhead.

Since ΔE_{scalar} contributes to the estimated cost ΔE of an edge collapse, we obtain simplified meshes whose faces naturally adapt to the attribute fields, as shown in Figures 8 and 11.

Optimizing scalar attributes at corners Our scheme for optimizing the scalar corner attributes S is a straightforward generalization of the scheme just described. Instead of solving for a single unknown attribute value $\underline{\mathbf{v}}_s$, the algorithm partitions the corners around v_s into continuous sets (based on equivalence of corner attributes) and for each continuous set solves independently for its optimal attribute value.

Range constraints Some scalar attributes have constrained ranges. For instance, the components (r, g, b) of color are typically constrained to lie between 0 and 1. Least-squares optimization may yield color values outside this range. In these cases we clip the optimized values to the given range. For least-squares minimization of a Euclidean norm at a single vertex, this is in fact optimal.

Normals Surface normals (n_x, n_y, n_z) are typically constrained to have unit length, and thus their domain is non-Cartesian. Optimizing over normals would therefore require minimization of a nonlinear functional with nonlinear constraints. We decided to instead simply carry the normals through the simplification process. Specifically, we compute the new normals at vertex $v_{s_i}^i$ by interpolating between the normals at vertices $v_{s_i}^{i+1}$ and $v_{m_0+i+1}^{i+1}$ using the α value that resulted in the best vertex position $\mathbf{v}_{s_i}^i$ in Section 4.3. Fortunately, the absolute directions of normals are less visually important than their discontinuities, and we have a scheme for preserving such discontinuities, as described in the next section.

4.5 Preserving discontinuity curves (E_{disc})

Appearance attributes give rise to a set of discontinuity curves on the mesh, both from differences between discrete face attributes (e.g. material boundaries), and from differences between scalar corner attributes (e.g. creases and shadow boundaries). As these discontinuity curves form noticeable features, we have found it useful to preserve them both topologically and geometrically.

We can detect when a candidate edge collapse would modify the topology of the discontinuity curves using some simple tests on the presence of sharp edges in its neighborhood. Let $sharp(v_j, v_k)$ denote that an edge $\{v_j, v_k\}$ is sharp, and let $\#sharp(v_j)$ be the number of sharp edges adjacent to a vertex v_j. Then, referring to Figure 1, $ecol(\{v_s, v_t\})$ modifies the topology of discontinuity curves if either:

- $sharp(v_s, v_l)$ and $sharp(v_t, v_l)$, or
- $sharp(v_s, v_r)$ and $sharp(v_t, v_r)$, or
- $\#sharp(v_s) \geq 1$ and $\#sharp(v_t) \geq 1$ and not $sharp(v_s, v_t)$, or
- $\#sharp(v_s) \geq 3$ and $\#sharp(v_t) \geq 3$ and $sharp(v_s, v_t)$, or
- $sharp(v_s, v_t)$ and $\#sharp(v_s) = 1$ and $\#sharp(v_t) \neq 2$, or
- $sharp(v_s, v_t)$ and $\#sharp(v_t) = 1$ and $\#sharp(v_s) \neq 2$.

If an edge collapse would modify the topology of discontinuity curves, we either disallow it, or penalize it as discussed in Section 4.6.

To preserve the geometry of the discontinuity curves, we sample an additional set of points X_{disc} from the sharp edges of $\hat{M}$, and define an additional energy term E_{disc} equal to the total squared distances of each of these points to the discontinuity curve from which it was sampled. Thus E_{disc} is defined just like E_{dist}, except that the points X_{disc} are constrained to project onto a set of sharp edges in the mesh. In effect, we are solving a curve fitting problem embedded within the surface fitting problem. Since all boundaries of the surface are defined to be discontinuity curves, our procedure preserves bound-

[3] The neighborhood of an edge collapse transformation is the set of faces shown in Figure 1. Using C notation, we set $\kappa = r < 4 ? 10^{-2} : r < 8 ? 10^{-4} : 10^{-8}$ where r is the ratio of the number of points to faces in the neighborhood.

ary geometry more accurately than [9]. Figure 9 demonstrates the importance of using the E_{disc} energy term in preserving the material boundaries of a mesh with discrete face attributes.

4.6 Permitting changes to topology of discontinuity curves

Some meshes contain numerous discontinuity curves, and these curves may delimit features that are too small to be visible when viewed from a distance. In such cases we have found that strictly preserving the topology of the discontinuity curves unnecessarily curtails simplification. We have therefore adopted a hybrid strategy, which is to permit changes to the topology of the discontinuity curves, but to penalize such changes. When a candidate edge collapse $ecol(\{v_s, v_t\})$ changes the topology of the discontinuity curves, we add to its cost ΔE the value $|X_{disc,\{v_s,v_t\}}| \cdot \|\mathbf{v}_s - \mathbf{v}_t\|^2$ where $|X_{disc,\{v_s,v_t\}}|$ is the number of points of X_{disc} projecting onto $\{v_s, v_t\}$. That simple strategy, although ad hoc, has proven very effective. For example, it allows the dark gray window frames of the "cessna" (visible in Figure 9) to vanish in the simplified meshes (Figures 5a–c).

Table 1: Parameter settings and quantitative results.

Object	Original $\hat{M}$		Base M^0		User param.		$\lvert X_{disc}\rvert$	V	Time
	m_0+n	#faces	m_0	#faces	$\lvert X\rvert-(m_0+n)$	c_{color}		$\frac{\text{bits}}{n}$	mins
cessna	6,795	13,546	97	150	100,000	-	46,811	46	23
terrain	33,847	66,960	3	1	0	-	3,796	46	16
mandrill	40,000	79,202	3	1	0	0.1	4,776	31	19
radiosity	78,923	150,983	1,192	1,191	200,000	0.01	74,316	37	106
fandisk	6,475	12,946	27	50	10,000	-	5,924	50	19

5 RESULTS

Table 1 shows, for the meshes in Figures 5–12, the number of vertices and faces in both $\hat{M}$ and M^0. In general, we let the simplification proceed until no more legal edge collapse transformations are possible. For the "cessna", we stopped at 150 faces to obtain a visually aesthetic base mesh. As indicated, the only user-specified parameters are the number of additional points (besides the $m_0 + n$ vertices of $\hat{M}$) sampled to increase fidelity, and the c_{scalar} constants relating the scalar attribute accuracies to the geometric accuracy. The only scalar attribute we optimized is color, and its c_{scalar} constant is denoted as c_{color}. The number $|X_{disc}|$ of points sampled from sharp edges is set automatically so that the densities of X and X_{disc} are proportional.[4] Execution times were obtained on a 150MHz Indigo2 with 128MB of memory.

Construction of the PM representation proceeds in three steps. First, as the simplification algorithm applies a sequence $ecol_{n-1} \ldots ecol_0$ of transformations to the original mesh, it writes to a file the sequence $vsplit_{n-1} \ldots vsplit_0$ of corresponding inverse transformations. When finished, the algorithm also writes the resulting base mesh M^0. Next, we reverse the order of the *vsplit* records. Finally, we renumber the vertices and faces of $(M^0, vsplit_0 \ldots vsplit_{n-1})$ to match the indexing scheme of Section 3.1 in order to obtain a concise format.

Figure 6 shows a single geomorph between two meshes M^{175} and M^{425} of a PM representation. For interactive LOD, it is useful to select a sequence of meshes from the PM representation, and to construct successive geomorphs between them. We have obtained good results by selecting meshes whose complexities grow exponentially, as in Figure 5. During execution, an application can adjust the granularity of these geomorphs by sampling additional meshes from the PM representation, or freeing some up.

In Figure 10, we selectively refined a terrain (grid of 181×187 vertices) using a new REFINE(v) function that keeps more detail near silhouette edges and near the viewer. More precisely, for the faces F_v adjacent to v, we compute the signed projected screen areas $\{a_f : f \in F_v\}$. We let REFINE(v) return TRUE if

(1) any face $f \in F_v$ lies within the view frustum, and either

(2a) the signs of a_f are not all equal (i.e. v lies near a silhouette edge) or

(2b) $\sum_{f\in F_v} a_f > thresh$ for a screen area threshold $thresh = 0.16^2$ (where total screen area is 1).

6 RELATED WORK

Mesh simplification methods A number of schemes construct a discrete sequence of approximating meshes by repeated application of a simplification procedure. Turk [17] sprinkles a set of points on a mesh, with density weighted by estimates of local curvature, and then retriangulates based on those points. Both Schroeder et al. [14] and Cohen et al. [4] iteratively remove vertices from the mesh and retriangulate the resulting holes. Cohen et al. are able to bound the maximum error of the approximation by restricting it to lie between two offset surfaces. Hoppe et al. [9] find accurate approximations through a general mesh optimization process (Section 4.1). Rossignac and Borrel [12] merge vertices of a model using spatial binning. A unique aspect of their approach is that the topological type of the model may change in the process. Their method is extremely fast, but since it ignores geometric qualities like curvature, the resulting approximations can be far from optimal. Some of the above methods [12, 17] permit the construction of geomorphs between successive simplified meshes.

Multiresolution analysis (MRA) Lounsbery et al. [10, 11] generalize the concept of multiresolution analysis to surfaces of arbitrary topological type. Eck et al. [7] describe how MRA can be applied to the approximation of an arbitrary mesh. Certain et al. [2] extend MRA to capture color, and present a multiresolution Web viewer supporting progressive transmission. MRA has many similarities with the PM scheme, since both store a simple base mesh together with a stream of detail records, and both produce a continuous-resolution representation. It is therefore worthwhile to highlight their differences:

Advantages of PM over MRA:

- MRA requires that the detail terms (wavelets) lie on a domain with subdivision connectivity, and as a result an arbitrary initial mesh $\hat{M}$ can only be recovered to within an ϵ tolerance. In contrast, the PM representation is lossless since $M^n = \hat{M}$.
- Because the approximating meshes $M^i, i < n$ in a PM may have arbitrary connectivity, they can be much better approximations than their MRA counterparts (Figure 12).
- The MRA representation cannot deal effectively with surface creases, unless those creases lie parametrically along edges of the base mesh (Figure 12). PM's can introduce surface creases anywhere and at any level of detail.
- PM's capture continuous, piecewise-continuous, and discrete appearance attributes. MRA schemes can represent discontinuous functions using a piecewise-constant basis (such as the Haar basis as used in [2, 13]), but the resulting approximations have too many discontinuities since none of the basis functions meet continuously. Also, it is not clear how MRA could be extended to capture discrete attributes.

[4] We set $|X_{disc}|$ such that $|X_{disc}|/perim = c(|X|/area)^{\frac{1}{2}}$ where $perim$ is the total length of all sharp edges in $\hat{M}$, $area$ is total area of all faces, and the constant $c = 4.0$ is chosen empirically.

Advantages of MRA over PM:

- The MRA framework provides a parametrization between meshes at various levels of detail, thus making possible multiresolution surface editing. PM's also offer such a parametrization, but it is not smooth, and therefore multiresolution editing may be non-intuitive.
- Eck et al. [7] construct MRA approximations with guaranteed maximum error bounds to $\hat{M}$. Our PM construction algorithm does not provide such bounds, although one could envision using simplification envelopes [4] to achieve this.
- MRA allows geometry and color to be compressed independently [2].

Other related work There has been relatively little work in simplifying arbitrary surfaces with functions defined over them. One special instance is image compression, since an image can be thought of as a set of scalar color functions defined on a quadrilateral surface. Another instance is the framework of Schröder and Sweldens [13] for simplifying functions defined over the sphere. The PM representation, like the MRA representation, is a generalization in that it supports surfaces of arbitrary topological type.

7 SUMMARY AND FUTURE WORK

We have introduced the progressive mesh representation and shown that it naturally supports geomorphs, progressive transmission, compression, and selective refinement. In addition, as a PM construction method, we have presented a new mesh simplification procedure designed to preserve not just the geometry of the original mesh, but also its overall appearance.

There are a number of avenues for future work, including:

- Development of an explicit metric and optimization scheme for preserving surface normals.
- Experimentation with PM editing.
- Representation of articulated or animated models.
- Application of the work to progressive subdivision surfaces.
- Progressive representation of more general simplicial complexes (not just 2-d manifolds).
- Addition of spatial data structures to permit efficient selective refinement.

We envision many practical applications for the PM representation, including streaming of 3D geometry over the Web, efficient storage formats, and continuous LOD in computer graphics applications. The representation may also have applications in finite element methods, as it can be used to generate coarse meshes for multigrid analysis.

ACKNOWLEDGMENTS

I wish to thank Viewpoint Datalabs for providing the "cessna" mesh, Pratt & Whitney for the gas turbine engine component ("fandisk"), Softimage for the "terrain" mesh, and especially Steve Drucker for creating several radiosity models using Lightscape. Thanks also to Michael Cohen, Steven "Shlomo" Gortler, and Jim Kajiya for their enthusiastic support, and to Rick Szeliski for helpful comments on the paper. Mark Kenworthy first coined the term "geomorph" in '92 to distinguish them from image morphs.

REFERENCES

[1] APPLE COMPUTER, INC. *3D graphics programming with QuickDraw 3D*. Addison Wesley, 1995.

[2] CERTAIN, A., POPOVIC, J., DUCHAMP, T., SALESIN, D., STUETZLE, W., AND DEROSE, T. Interactive multiresolution surface viewing. *Computer Graphics (SIGGRAPH '96 Proceedings)* (1996), 91–98.

[3] CLARK, J. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM 19*, 10 (October 1976), 547–554.

[4] COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., BROOKS, F., AND WRIGHT, W. Simplification envelopes. *Computer Graphics (SIGGRAPH '96 Proceedings)* (1996), 119–128.

[5] CURLESS, B., AND LEVOY, M. A volumetric method for building complex models from range images. *Computer Graphics (SIGGRAPH '96 Proceedings)* (1996), 303–312.

[6] DEERING, M. Geometry compression. *Computer Graphics (SIGGRAPH '95 Proceedings)* (1995), 13–20.

[7] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution analysis of arbitrary meshes. *Computer Graphics (SIGGRAPH '95 Proceedings)* (1995), 173–182.

[8] FUNKHOUSER, T., AND SÉQUIN, C. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (SIGGRAPH '93 Proceedings)* (1993), 247–254.

[9] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Mesh optimization. *Computer Graphics (SIGGRAPH '93 Proceedings)* (1993), 19–26.

[10] LOUNSBERY, J. M. *Multiresolution analysis for surfaces of arbitrary topological type*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1994.

[11] LOUNSBERY, M., DEROSE, T., AND WARREN, J. Multiresolution analysis for surfaces of arbitrary topological type. Submitted for publication. (TR 93-10-05b, Dept. of Computer Science and Engineering, U. of Washington, January 1994.).

[12] ROSSIGNAC, J., AND BORREL, P. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics*, B. Falcidieno and T. L. Kunii, Eds. Springer-Verlag, 1993, pp. 455–465.

[13] SCHRÖDER, P., AND SWELDENS, W. Spherical wavelets: efficiently representing functions on the sphere. *Computer Graphics (SIGGRAPH '95 Proceedings)* (1995), 161–172.

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

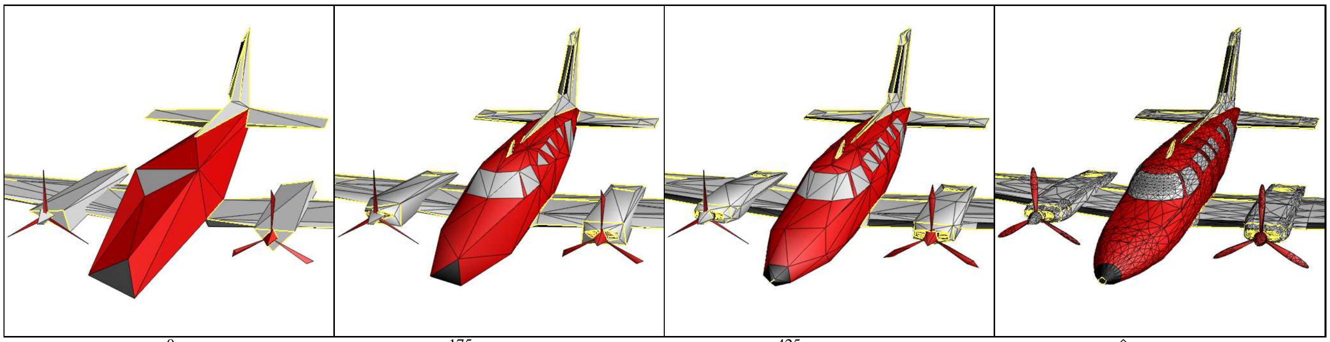
[15] TAUBIN, G., AND ROSSIGNAC, J. Geometry compression through topological surgery. Research Report RC-20340, IBM, January 1996.

[16] TURAN, G. Succinct representations of graphs. *Discrete Applied Mathematics 8* (1984), 289–294.

[17] TURK, G. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings) 26*, 2 (1992), 55–64.

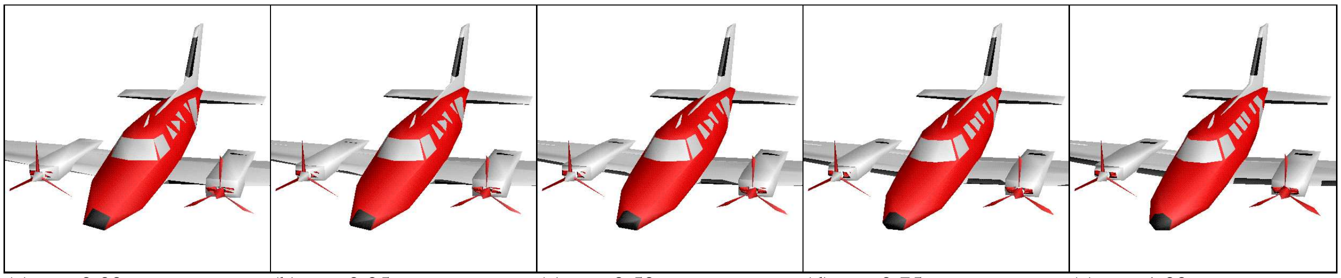
[18] UPSTILL, S. *The RenderMan Companion*. Addison-Wesley, 1990.

[19] WITTEN, I., NEAL, R., AND CLEARY, J. Arithmetic coding for data compression. *Communications of the ACM 30*, 6 (June 1987), 520–540.



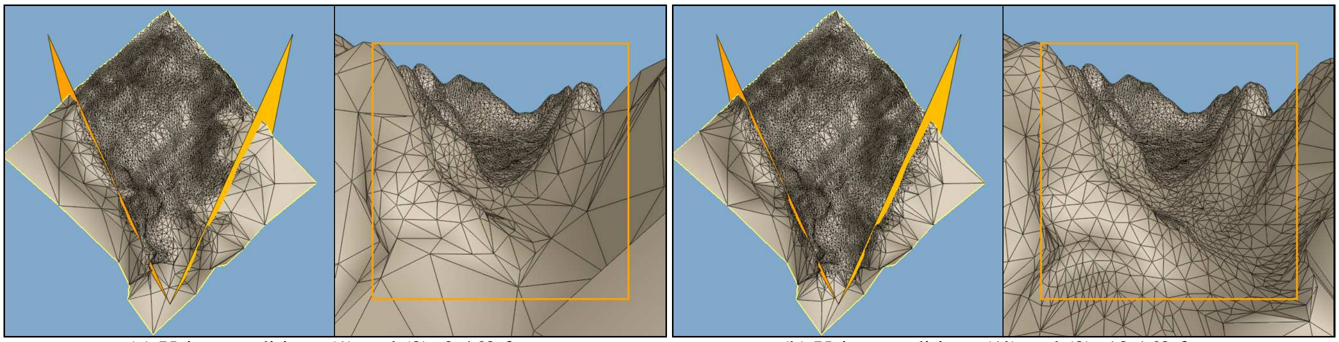
(a) Base mesh M^0 (150 faces) (b) Mesh M^{175} (500 faces) (c) Mesh M^{425} (1,000 faces) (d) Original $\hat{M}=M^n$ (13,546 faces)

Figure 5: The PM representation of an arbitrary mesh $\hat{M}$ captures a continuous-resolution family of approximating meshes $M^0 \ldots M^n=\hat{M}$.



(a) α = 0.00 (b) α = 0.25 (c) α = 0.50 (d) α = 0.75 (e) α = 1.00

Figure 6: Example of a geomorph $M^G(\alpha)$ defined between $M^G(0)\doteq M^{175}$ (with 500 faces) and $M^G(1)=M^{425}$ (with 1,000 faces).



(a) Using conditions (1) and (2); 9,462 faces (b) Using conditions (1') and (2); 12,169 faces

Figure 7: Example of selective refinement within the view frustum (indicated in orange).

(a) $\hat{M}$ (200×200 vertices) (b) Simplified mesh (400 vertices)

Figure 8: Demonstration of minimizing E_{scalar}: simplification of a mesh with trivial geometry (a square) but complex scalar attribute field. ($\hat{M}$ is a mesh with regular connectivity whose vertex colors correspond to the pixels of an image.)

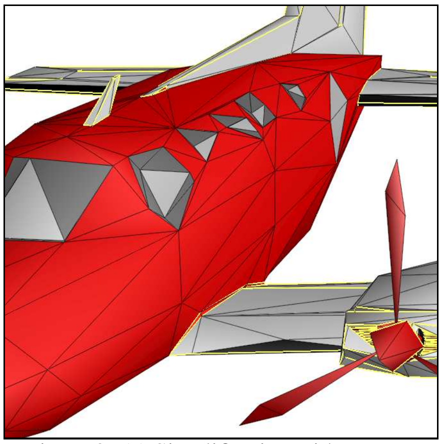


Figure 9: (a) Simplification without E_{disc}

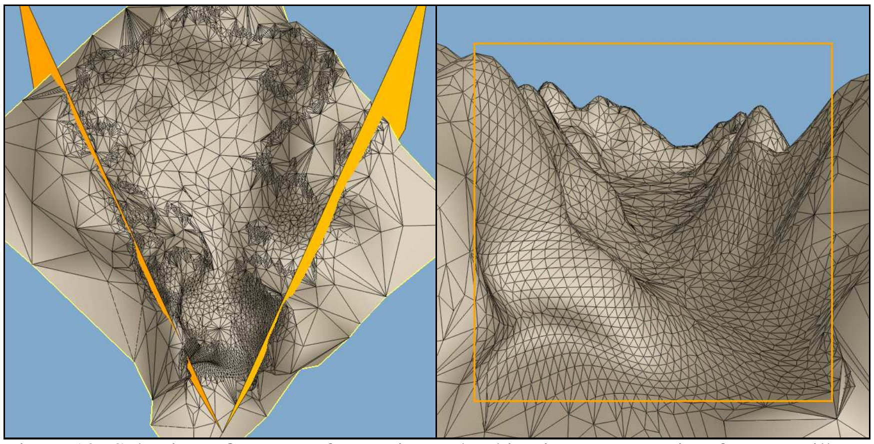


Figure 10: Selective refinement of a terrain mesh taking into account view frustum, silhouette regions, and projected screen size of faces (7,438 faces).

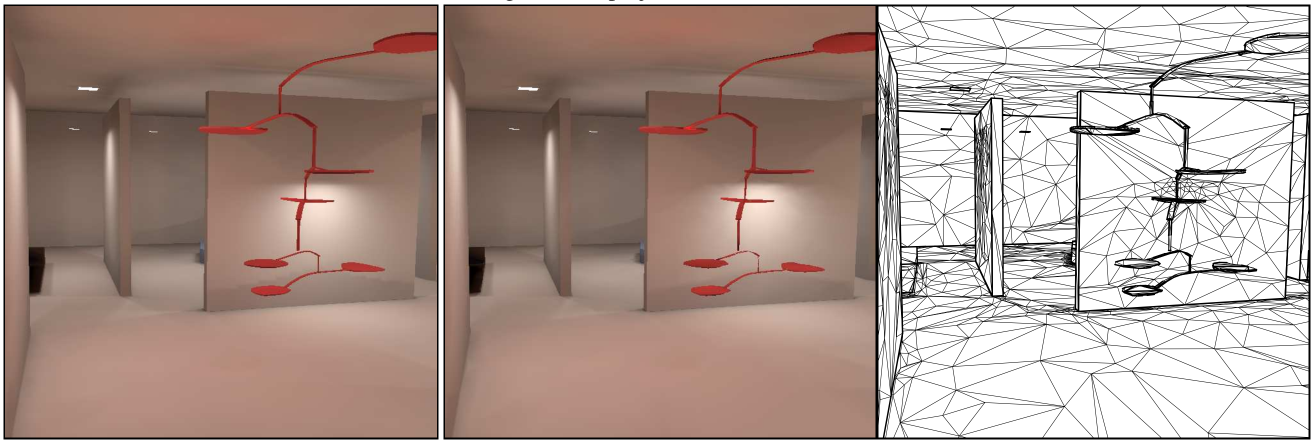
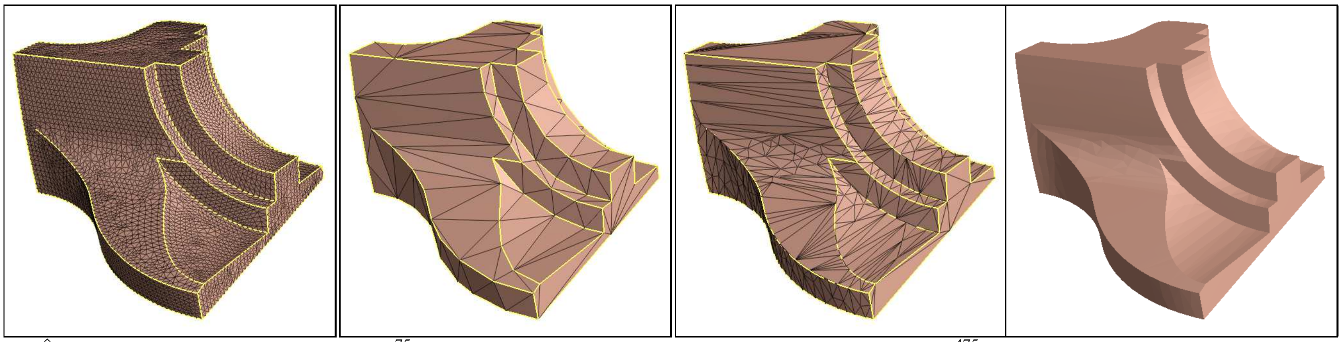
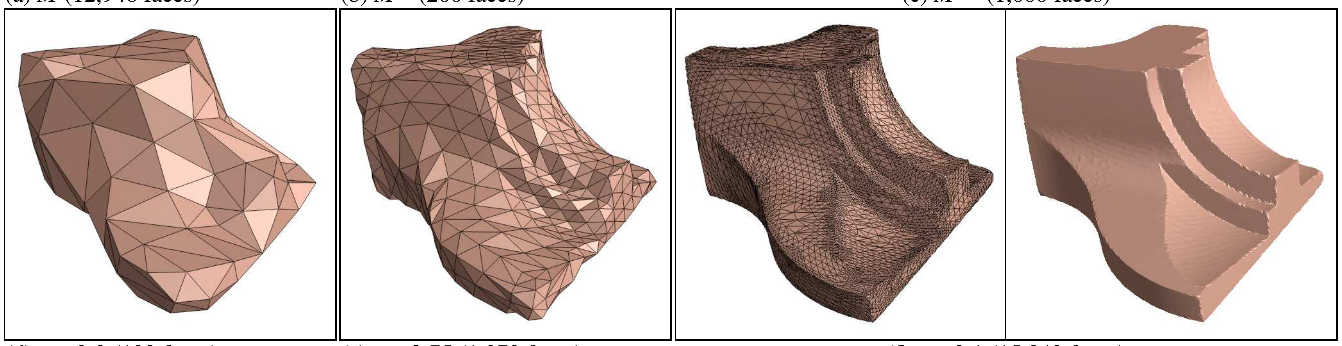


Figure 11: Simplification of a radiosity solution; left: original mesh (150,983 faces); right: simplified mesh (10,000 faces).



(a) $\hat{M}$ (12,946 faces) (b) M^{75} (200 faces) (c) M^{475} (1,000 faces)



(d) $\epsilon = 9.0$ (192 faces) (e) $\epsilon = 2.75$ (1,070 faces) (f) $\epsilon = 0.1$ (15,842 faces)

Figure 12: Approximations of a mesh $\hat{M}$ using (b–c) the PM representation, and (d–f) the MRA scheme of Eck et al. [7]. As demonstrated, MRA cannot recover $\hat{M}$ exactly, cannot deal effectively with surface creases, and produces approximating meshes of inferior quality.