View-dependent refinement of multiresolution meshes with subdivision connectivity

Daniel I. Azuma[1] *Brian Curless*[1] *Tom Duchamp*[1]
David H. Salesin[1,2] *Werner Stuetzle*[1] *Daniel N. Wood*[1]

[1]University of Washington [2]Microsoft Research

Abstract

We develop a new view-dependent level-of-detail algorithm for triangle meshes with subdivision connectivity. The algorithm is more suitable for textured meshes of arbitrary topology than existing progressive mesh-based schemes. It begins with a wavelet decomposition of the mesh, and, per frame, finds a partial sum of wavelets necessary for high-quality renderings from that frame's viewpoint. We present a new screen-space error metric that measures both geometric and texture deviation. In addition, wavelets that lie outside the view frustum or in backfacing areas are eliminated. The algorithm takes advantage of frame-to-frame coherence for improved performance, and supports geomorphs for smooth transitions between levels of detail.

1 Introduction

Complex graphical environments consisting of thousands or millions of polygons are becoming commonplace. These environments arise in computer-aided design applications, visualizations of scientific data sets, and 3D photography techniques such as laser scanning. Current hardware, however, is not capable of rendering many of these large datasets at sufficiently high frame rate for interactive applications.

While scene complexity increases with the power of acquisition and modeling tools, the display resolution is growing slowly and will someday reach a limit imposed by the needs of the human visual system. As a result of this mismatch in complexity and resolution growth, practioners often find that many polygons are being rendered, unnecessarily, to a single pixel. Further, many polygons are not visible to the user because they lie outside the viewing frustum, or they are back-facing, or they are occluded by other polygons. In such cases, the scene can be rendered with far fewer polygons with no appreciable effect on the final rendered image.

Many researchers have proposed methods for reducing the complexity of meshes (also known as mesh decimation) using such schemes as re-triangulation [18], vertex removals [16], edge collapses [9], and vertex clustering [15]. By constructing a set of approximating meshes, an appropriate *level of detail* (LOD) can be selected based on the current viewpoint [6, 5, 3].

More recently, researchers have developed methods for continuous transformations between levels of detail, the most prevalent of which is the progressive mesh method introduced by Hoppe [7]. Such transformations enable a more powerful form of LOD construction that allow the mesh simplification to vary *over the surface* of an object, selecting a coarse level of detail for invisible regions of an object, and finer detail for visible areas, particularly along silhouettes. For instance, when rendering a fly-over of a terrain, these systems can finely tessellate regions close to the camera and coarsely tessellate areas far away from the camera or outside its view frustum. Xia and Varshney [20] and Hoppe [8][10] describe view-dependent LOD frameworks built atop modifications of Hoppe's progressive mesh representation. Luebke and Erikson [14] describe an alternate method based on a vertex clustering simplification algorithm that can change the topology of the triangle mesh, though the progressive mesh framework tends to yield higher quality renderings.

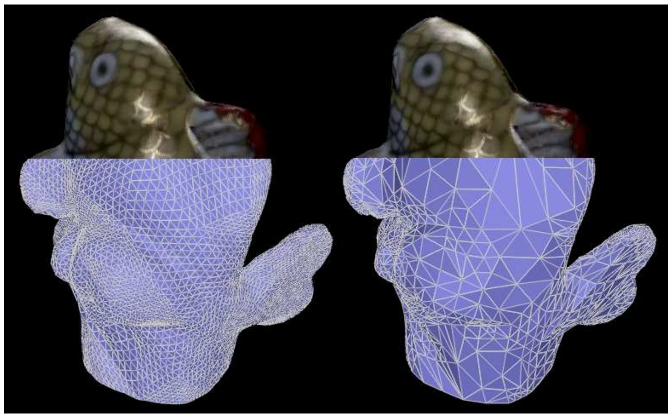
Figure 1 View-dependent refinement of a base mesh using a parametric error metric ensures that the simplified mesh (right) not only approximates the high resolution geometry (left) but also that surface texture is not distorted.

The progressive mesh approach does, however, have some shortcomings. First, the algorithm is intended to work on static meshes and, because it does not in general yield a smooth mapping among the levels in the hierarchy, is not immediately suited to such operations as mesh editing, signal processing over surfaces, and animation. Second, texture-mapping is only supported when the surface is parameterized prior to simplification, a separate, complex step in itself. Third, during rendering of texture-mapped surfaces, additional machinery is necessary to minimize texture distortion [3]. Finally, during view-dependent refinement, coarsening operations are difficult to implement due to dependencies in the hierarchy [10].

An attractive alternative to progressive meshes is to represent a surface in a multiresolution framework based on four-to-one subdivisions, such as the one described by Lounsbery *et al.* [13]. Meshes of this type have a restricted connectivity known as *subdivision connectivity*. These multiresolution meshes have been demonstrated to admit powerful editing operations such as those described by Zorin *et al.* [21]. Further, when starting with high resolution geometry, Eck *et al.* [5] and Lee *et al.* [11] have demonstrated methods for constructing the subdivision hierarchies while simultaneously building low-distortion parameterizations which can be tuned to respect prevalent geometric features during construction.

Several researchers have demonstrated methods for efficient rendering of these multiresolution meshes. Certain *et al.* [2] demonstrate progressive refinement of meshes and texture maps independently, though viewpoint is not taken into account. Zorin *et al.* [21] demonstrate view-*independent* level of detail. Wood *et al.* [19] introduce a framework for continuous, view-dependent LOD of subdivision connectivity meshes, but provide only sparse details or analysis.

In this paper, we describe in detail a method that fills the gap in this important area of geometric representation and modeling. In particular, our contributions are:

- A view-dependent refinement scheme with detail varying over the surface of multiresolution subdivision-connectivity surfaces. The scheme includes adaptive simplification based on:
  - Frustum culling
  - Backface culling
  - A screen space error metric with inherent texture distortion reduction
- An improved metric for screen space error
- Support of overlapping, coarsening, run-time geomorphs

The remainder of this paper is organized as follows. We begin by reviewing the technique of [8], which is most similar to our algorithm. Next, we summarize the mathematical basis of mesh parameterization and multiresolution analysis and describe how this analysis leads to a view-dependent algorithm. We then present the qualitative and quantitative results of our implementation and conclude with a discussion of future work.

2 View-dependent LOD for progressive meshes

A progressive mesh [7] is an ordered family of approximations of a triangle mesh represented by a coarse base mesh and a sequence of *vertex split* operations. The sequence is constructed by repeatedly applying the inverse operation, *edge collapse*, starting with the original, highest resolution, mesh. Consequently, starting with the base mesh and applying the entire stream of vertex splits in order will yield the original mesh.

Xia and Varshney [20] and Hoppe [8] refined the progressive mesh representation to support selective level of detail. Vertices in the mesh are organized into a forest of trees in which each tree root is a vertex in the base mesh and the children of any given vertex are the vertices formed by applying the vertex split operation to it. A cut through the forest represents a possible refinement of the mesh. In particular, the cut through the root vertices of the forest represents the base mesh, and the cut through the leaf nodes represents the original, highest resolution mesh.

Hoppe's view-dependent, LOD algorithm operates incrementally from frame to frame. The mesh in each frame is represented as a list of active vertices along a cut through the forest. Prior to rendering a frame from a new viewpoint, the active list of vertices is traversed, and a refinement function is evaluated to query whether each vertex in the list should be split or its parent edge collapsed, based on the new viewpoint. This process moves the cut up or down, and the mesh is incrementally modified by performing the corresponding vertex split and edge collapse operations. Finally, the modified mesh is rendered.

The refinement function selects a desired refinement state for a vertex based on three view-dependent criteria. First, areas of the mesh that lie completely outside the view frustum are coarsened. Second, areas of the mesh that face away from the viewer are also coarsened. Finally, a screen-space error bound is enforced.

To reduce the "popping" effect of switching between levels of detail, Hoppe [7] introduces geomorphs, a technique for smoothly interpolating between successive meshes in a progressive mesh hierarchy. Hoppe [8] notes that geomorphs can be used to smooth transitions in a view-dependent level-of-detail framework and in later work [10] he first incorporates *run-time* geomorphs in the context of terrain fly-throughs. Geomorphs of vertex splits are handled in a straightforward manner. Geomorphs of edge collapses, however, present greater difficulty, as "overlaps" or dependencies can arise and require a parent to be geomorphed at the same time as its child. To avoid such difficulties, he requires the children to be geomorphed and collapsed before beginning the geomorph of the parent.

3 View-dependent LOD for subdivision-connectivity meshes

In this section, we describe a new algorithm for view-dependent level-of-detail of triangle meshes with subdivision connectivity. Our method relies on multiresolution analysis of triangle meshes [12, 17, 2], ensuring a smooth mapping between meshes generated at different levels of detail. We describe a screen-space error metric that measures texture deviation as well as geometric deviation. Like the earlier work on progressive mesh schemes [8], our algorithm operates incrementally, taking advantage of temporal coherence, and it supports geomorphs for smooth transitions between meshes generated in different frames, including overlapping coarsening geomorphs.

3.1 Decoupling color from geometry

We define a *colored mesh* to be a triangular mesh $M \subset \mathbb{R}^3$, together with an RGB-valued function $M \to Color$.

Representing a colored mesh as a textured surface requires a parameterization of M. For acquired surfaces with arbitrary topology, a parameterization is typically generated using a scheme such as those described in Eck *et al.* [5] or Lee *et al.* [11]. Those schemes construct a homeomorphism (or *parameterization*)

$$\boldsymbol{\rho} : K^0 \to M \subset \mathbb{R}^3 ,$$

where the *base complex* K^0 is an abstract simplicial surface, consisting of a relatively small number of faces. The composition

$$c_{RGB} : K^0 \xrightarrow{\boldsymbol{\rho}} M \longrightarrow Color$$

transfers the color function to the base complex K^0, whose faces can then be viewed as texture domains [2]. Notice that we can texture map the image of any map $\boldsymbol{\rho}_{approx} : K^0 \to \mathbb{R}^3$ approximating $\boldsymbol{\rho}$, provided only that it is piecewise linear with respect to a refinement of the triangulation of K^0. Furthermore, although we discuss only colored meshes, the extension to other surface properties (*e.g.*, BRDF's, normals, transparency) is straightforward.

3.2 Multiresolution Analysis

Our construction of the approximation $\boldsymbol{\rho}_{approx}$ is based on multiresolution analysis. Let K^j denote the triangulation of K^0 obtained by applying j four-to-one subdivisions of each face of K^0. Notice that the vertices of K^j consist of the vertices of K^{j-1} together with additional vertices located at the midpoints of the edges of K^{j-1}. These vertices are called *edge vertices* at level j. The vertices of K^j are arranged in the following hierarchy: v_i^0 denotes a vertex of K^0 and v_e^j denotes the edge vertex at level j centered on the edge e of K^{j-1}. Points of M are indicated with bold face: $\mathbf{v}_a^j = \boldsymbol{\rho}(v_a^j)$.

For each integer j we let

$$\boldsymbol{\rho}^j : K^0 \longrightarrow \mathbb{R}^3$$

be the unique piecewise linear map with respect to the triangulation K^j such that $\boldsymbol{\rho}^j(v) = \boldsymbol{\rho}(v)$ for each vertex v of K^j. Because the sequence $\boldsymbol{\rho}^j$ converges uniformly to $\boldsymbol{\rho}$, the sequence of increasingly fine meshes $M^j = \boldsymbol{\rho}^j(K^0) \subset \mathbb{R}^3$ converges to M (see Figure 2).

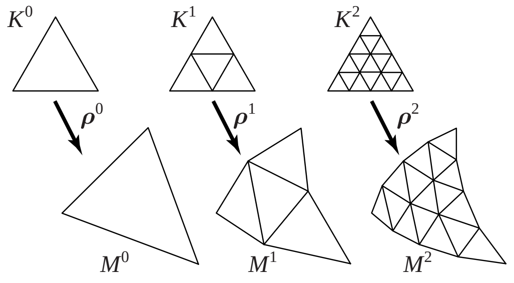

Figure 2 To generate successive levels of detail, apply four-to-one subdivisions and perturb the new vertices to their final positions.

The map $\boldsymbol{\rho}^0$ is the coarsest level approximation of $\boldsymbol{\rho}$. For each integer $j > 0$, we view $\boldsymbol{\rho}^j$ as a perturbation of its predecessor $\boldsymbol{\rho}^{j-1}$. More formally, $\boldsymbol{\rho}^j$ has the expansion

$$\boldsymbol{\rho}^j = \boldsymbol{\rho}^{j-1} + \sum_e \boldsymbol{s}_e^j \, \widehat{\psi}_e^{\,j}, \tag{1}$$

where the summation is over the edges of K^j, $\widehat{\psi}_e^{\,j}$ denotes the piecewise linear "hat function" that assumes the value 1 at the edge vertex v_e^j and the value 0 at every other vertex of K^j (Figure 3).

Repeated application of (1) yields the *lazy wavelet* expansion [2]

$$\boldsymbol{\rho} = \boldsymbol{\rho}^0 + \sum_{j=1}^{\infty} \sum_{e \in \mathrm{Edges}(K^{j-1})} \boldsymbol{s}_e^j \, \widehat{\psi}_e^{\,j} \, ,$$

Other wavelet bases, such as the k-disk wavelets [17], are more nearly orthogonal; however, for the purposes of a view-dependent level-of-detail scheme, orthogonality is less important than the speed offered by the simplicity of the lazy wavelet expansion. As rendering speed is dependent on the number of triangles in an expansion and not the number of wavelets, the simple triangulation of a lazy wavelet is desirable.

In practice, we truncate the infinite sum at level J, where J is chosen so that the error $|\boldsymbol{\rho} - \boldsymbol{\rho}^J|$ is below a user-specified threshold. In other words we assume that $\boldsymbol{\rho} = \boldsymbol{\rho}^J$ and, consequently, that $M = M^J$.

3.3 Adaptation criteria

Our goal is the following: at each frame t, we wish to find an approximation $\boldsymbol{\rho}_t$ to $\boldsymbol{\rho}$ that generates a convincing image given the viewpoint parameters. We call the process of computing $\boldsymbol{\rho}_t$ the *adaptation stage*. In addition, we need an efficient method of rendering the resulting mesh, a process we call the *rendering stage*. We describe those two stages in more detail in the next two subsections.

We may now restate the problem in terms of the lazy wavelet expansion. At each frame t, we wish to find a small subset U_t of the index set

$$S = \{(e, j) \, : \, e \in \mathrm{Edges}(K^{j-1}), \; j \leq J\}$$

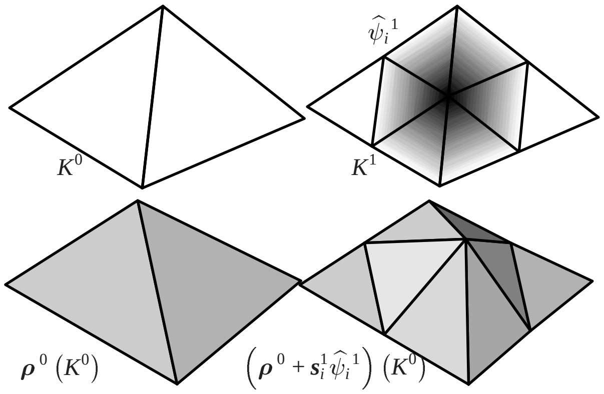
$\widehat{\psi}_i^{\,1}$
K^0
K^1
$\boldsymbol{\rho}^0\left(K^0\right)$
$\left(\boldsymbol{\rho}^0 + \boldsymbol{s}_i^1 \widehat{\psi}_i^{\,1}\right)\left(K^0\right)$

Figure 3 Adding a lazy wavelet. Top left: the base domain. Top right: domain after one subdivision, showing a hat function at level 1. Bottom left: the coarsest geometric mesh. Bottom right: the mesh with added correction term.

such that

$$\boldsymbol{\rho}_t = \boldsymbol{\rho}^0 + \sum_{(e,j) \in U_t} \boldsymbol{s}_e^j \, \widehat{\psi}_e^{\,j}$$

Our criteria for including an index (e, j) in U_t consists of three tests: (i) a *view-frustum test*, which excludes indices for which the support of $\widehat{\psi}_e^{\,j}$ is outside the view frustum, (ii) a *back-facing test*, which excludes indices for which the image of the support of the wavelet is back-facing, and (iii) a *screen-error test* that ensures small screen error in regions of high curvature or and along silhouettes, where high geometric detail is needed. Tests (i) and (ii) are nearly identical to those of Hoppe [8], with the exception that the areas of the mesh tested correspond to the support of our hat functions rather than the neighborhood of vertices and their descendants; test (iii) differs more substantially from the corresponding test described by Hoppe. Our test (iii) will implicitly measure not just geometric error but the deviation of surface properties (*e.g.,* texture distortion), and we employ a tighter bounding volume to measure screen space error.

3.3.1 View-frustum test

To test that the support of $\widehat{\psi}_e^{\,j}$ does not lie completely outside the view frustum, we pre-compute the radius r_e^j of the smallest sphere, centered around the point $\boldsymbol{v}_e^j \in M$ that bounds the $\boldsymbol{\rho}$-image of the support of $\widehat{\psi}_e^{\,j}$. We include the index (e, j) if any part of the sphere lies within the view frustum.

Notice that an index (e, j) satisfies the view-frustum test (and is included) if its support contains the support of a finer level wavelet $\widehat{\psi}_f^{\,k}$, and (f, k) satisfies the test.

3.3.2 Back-facing test

To determine if the image of the support of a wavelet is back-facing, we pre-compute the Gauss map (field of unit normals) of M over the support of each wavelet and find a bounding cone of normals centered around the normal to M at each point $\boldsymbol{v}_e^j$. We include (e, j) in U_t if the vector from $\boldsymbol{v}_e^j$ to the viewpoint makes an angle of less than $\pi/2$ with some vector in the cone.

Notice, again, that an index (e, j) is front-facing (and included in U_t) if its support contains the support of a finer level wavelet $\widehat{\psi}_f^{\,k}$, and (f, k) is front-facing.

3.3.3 Screen-error test

We want to approximate the map $\boldsymbol{\rho}$ sufficiently well that the error between the original map $\boldsymbol{\rho}$ and the approximation $\boldsymbol{\rho}_t$ at time t, projected into screen space, is bounded by a user-specified value. Our test is based on the *parametric error*, $\boldsymbol{\rho} - \boldsymbol{\rho}_t$. The parametric error does not measure the difference between a point on the approximate geometry and the closest point on the true geometry. Instead it measures the difference between a point on the approximation and the *corresponding* point on the true geometry. Therefore it accounts for the deviation of surface features like texture.

To pre-compute the parametric error, we have to make an assumption about the set U_t. Suppose that (f, k) is in U_t and that the support of $\widehat{\psi}_f^{\,k}$ is contained in the support of $\widehat{\psi}_e^{\,j}$. Then we also include its "ancestor" (e, j) in U_t.

We also will order our tests, so that we will not test (f, k) unless we have already included all of its ancestors. With these assumptions in place, we can pre-compute the parametric error that results from excluding (e, j) from U_t. At a point p in K^0 the error $\boldsymbol{E}_{param}(p)$ is given by

$$\boldsymbol{E}_{param}(p) = \boldsymbol{\rho}(p) - \boldsymbol{\rho}_t(p) = \boldsymbol{\rho}(p) - \boldsymbol{\rho}^{\,j-1}(p)\,. \qquad (2)$$

The error E resulting from excluding (e, j) is the union of the parametric errors for all p in the support of $\widehat{\psi}_e^{\,j}$. Because both $\boldsymbol{\rho}$ and $\boldsymbol{\rho}^{\,j-1}$ are piecewise linear on K^J, this set is contained in the convex hull of the set $\{\boldsymbol{E}_{param}(v)\}$ where v ranges over the vertices of K^J in the support of $\widehat{\psi}_e^{\,j}$. We use this set for following computation of a bounding box around the error E.

To accelerate this test, we bound E by a normal-aligned spheroid, and test the size of the spheroid's projection onto screen space. The spheroid is determined by the normal to M at $\boldsymbol{v}_e^j$, the length of its normal-aligned radius a_e^j, and the length b_e^j or its radius in the direction orthogonal to the normal. (Figure 4(b)).

Hoppe [8] uses as a bounding volume a sphere attached to two normal-aligned cones (Figure 4(a)). Most of the geometric deviation is likely to be normal to the surfaces; hence, the normal-aligned cones allow the volume to enclose longer error vectors while keeping a relatively small screen-space footprint. We could compute the optimal major axis but that would require extra storage if we are already storing normal information. The shape is also fairly cheap to project into screen space. A spheroid generally provides a tighter bounding volume around error vectors than Hoppe's shape. For example, consider the simple case of a single error vector deviating 15 degrees from the normal. As shown in Figure 4 the Hoppe-style bounding shape can be considerably larger than the spheroid. Our experiments (Section 4) demonstrate a smaller but consistent advantage for the spheroid.

We define the *screen space error* to be radius of the smallest circle containing the projection of the bounding spheroid onto the viewing screen, measured as a fraction of the size $2\cot(\varphi/2)$ of the viewport, where φ is the field-of-view angle. We want the screen space error to be less than a user-specified, *screen-space tolerance* τ. We ensure this by including the index (e, j) if the diameter of the screen-space projection of the bounding spheroid is greater than 2τ.

Our computation of a_e^j and b_e^j for a particular wavelet $\widehat{\psi}_i^{\,j}$ is similar to the computation in [8]. Given the set E of error vectors, we first determine the ratio using the expression

$$\frac{a_e^j}{b_e^j} = \frac{\max\limits_{\boldsymbol{e}_k \in E} \left(\boldsymbol{e}_k \cdot \hat{\mathbf{n}}_e^j\right)}{\max\limits_{\boldsymbol{e}_k \in E} \|\boldsymbol{e}_k \times \hat{\mathbf{n}}_e^j\|} \qquad (3)$$

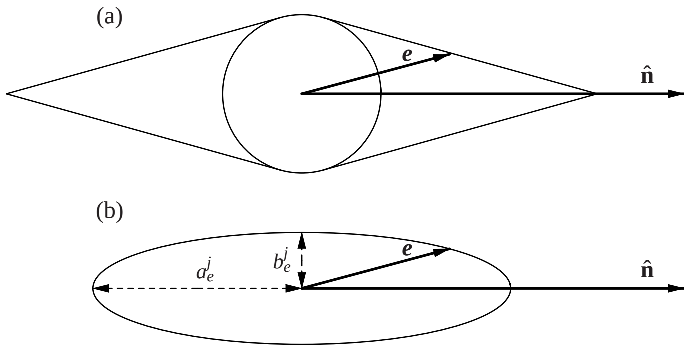

Figure 4 Bounding shapes constructed around a single error vector $\boldsymbol{e}$, pointing 15 degrees from the unit normal $\hat{\mathbf{n}}$. (a) Hoppe [8]. (b) Spheroid.

where $\hat{\mathbf{n}}_e^j$ is the unit normal at the vertex v_e^j. We then scale the spheroid so it bounds the vectors in E.

To test an index (e, j) for inclusion in U_t, we compare the diameter of the screen-space projection to the screen-space tolerance. More precisely, given the center of projection $\boldsymbol{q}_t$ at time t, the screen-space tolerance τ (as a fraction of viewport size), and the field-of-view angle φ, we include (e, j) and all of its ancestors in U_t if either

$$\alpha_e^j \|\boldsymbol{v}_e^j - \boldsymbol{q}_t\|^2 + \beta_e^j \left(\left(\boldsymbol{v}_e^j - \boldsymbol{q}_t\right) \cdot \hat{\mathbf{n}}_i\right)^2 > \kappa\, \|\boldsymbol{v}_e^j - \boldsymbol{q}_t\|^4 \qquad (4)$$

or

$$\left(b_e^j\right)^2 > \kappa\, \|\boldsymbol{v}_e^j - \boldsymbol{q}_t\|^2 \qquad (5)$$

where

$$\alpha_i^j = \left(a_e^j\right)^2, \quad \beta_i^j = \left(b_e^j\right)^2 - \left(a_e^j\right)^2, \quad \kappa = 4\tau^2 \cot^2\left(\frac{\varphi}{2}\right)\,.$$

The quantities α_i^j and β_i^j are pre-computed per wavelet, and κ is computed once per frame. With suitable combining of common subexpressions, the computation of expressions (4) and (5) requires one more floating-point multiply than the computation of the corresponding expression for Hoppe's bounding shape. The derivation of those expressions is in Appendix A.

For some applications, such as bump mapping or surface light fields, geometric or texture distortion in the interior of an object's image is often less noticeable than low-resolution silhouettes [19]. Therefore, it is sometimes desirable to use a lower tolerance near the silhouettes than in the interiors. We test whether a wavelet may contribute to the silhouette by examining the cone of normals we pre-computed for the back-facing test, and determining if it spans directions that are both front- and back-facing [14]. Any wavelet that may contribute to the silhouette is tested against the more stringent silhouette tolerance; others are tested against the less stringent interior tolerance.

3.4 Incremental adaptation

Because the set S can be very large, an exhaustive search of S, evaluating the inclusion criterion for each member, is not feasible. Instead, we exploit the hierarchical structure of wavelets to organize the set S into an acyclic, directed *dependency graph* G and use an incremental adaptation algorithm.

To give a formal definition of G, define an element $(j + 1, e') \in S$ to be a *child* of $(j, e) \in S$ if the support of $\widehat{\psi}_{e'}^{\,j+1}$ is completely contained in the support of $\widehat{\psi}_{e}^{\,j}$. In this case we say that $\widehat{\psi}_{e}^{\,j}$ is the *parent* of $\widehat{\psi}_{e'}^{\,j+1}$. The set S, together with the parent-child relation forms the directed graph G (see Figure 5). Vertices of G may have zero, one, or two parents, and up to six children. Observe that (f,k) is the descendent of (e,j) at a coarser level if and only if the support of $\widehat{\psi}_{f}^{\,k}$ is contained in the support of $\widehat{\psi}_{e}^{\,j}$.

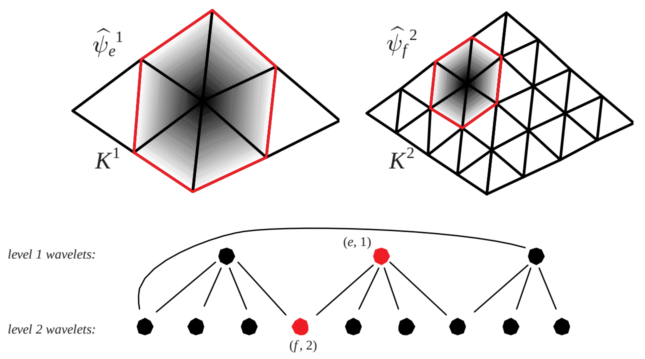

Figure 5 Wavelet parents and children. Wavelet e (top left) at level 1 is a parent of wavelet f (top right) at level 2. Bottom: a portion of the dependency graph G.

Recall that in formulating our screen-error test, we made the assumption that every ancestor of an element of U_t is also an element of U_t, we call this assumption the *closure condition*. As illustrated in Figure 3.4(a), U_t is completely determined by the (generally smaller) subset $C_t \subset U_t$ of all vertices of K^J that are either terminal vertices of G or have a child not contained in U_t. We call C_t the *cut set* of U_t.

Observe that the set of indices satisfying the view-frustum test also satisfy the closure condition. Suppose that the support of $\widehat{\psi}_{b}^{\,k}$ is contained in the support of $\widehat{\psi}_{a}^{\,j}$, and that (k,b) satisfied the view-frustum test. Then the bounding sphere centered at $\boldsymbol{v}_b^k$ intersects the view-frustum. Because the bounding sphere associated to $\widehat{\psi}_{b}^{\,k}$ is contained in the bounding sphere associated to $\widehat{\psi}_{a}^{\,j}$, (j,a) also satisfies the view-frustum test. A similar argument, based on the cone of normal, shows that the set of indices satisfying the back-facing test is closed.

This suggests an incremental algorithm for computing U_t: Begin with the cut defining U_{t-1}. Find U_t by traversing the cut C_{t-1}, moving it up or down according to the three adaptation tests for the view parameters. A wavelet is a candidate for addition to U_t only if both its parents are in U_t, and a wavelet is a candidate for removal from U_t only if none of its children are in U_t. In our implementation, we begin by setting U_0 to be the empty set—that is, $\boldsymbol{\rho}_0 = \boldsymbol{\rho}^{\,0}$.

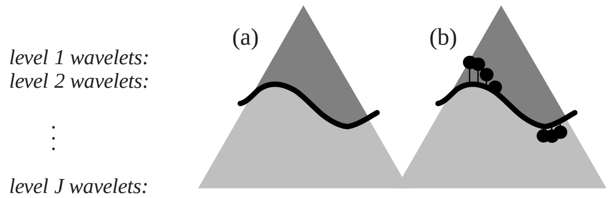

Figure 6 Left: The cut C_t (solid line) through the graph G separates wavelets in U_t (dark) from those not in U_t (light). Right: The adaptation stage moves the cut according to the view parameters of the next frame. Wavelets added and removed (shown with a line and dot) are geomorphed.

In the worst case, the above algorithm tests every wavelet in S. However, for an interactive application, the viewpoint typically does not change much from frame to frame. Consequently, $U_t \approx U_{t-1}$, implying that very few wavelets need to be tested per frame.

3.5 Runtime construction of the geomorph

To alleviate the "popping" effect that can accompany a transition between meshes at different levels of detail, Hoppe [7] proposes smoothly interpolating between the two meshes, a process he calls *geomorphing*. In [10] he demonstrates geomorphs constructed at runtime in a progressive mesh-based, view-dependent LOD framework. Our framework, based on the lazy wavelet decomposition, also supports geomorphs. To smoothly add or subtract a correction term $\boldsymbol{s}_e^j\,\widehat{\psi}_{e}^{\,j}$ between frames t_0 and t_n, we scale it according to the expressions:

$$\frac{t-t_0}{t_n-t_0}\boldsymbol{s}_e^j\,\widehat{\psi}_{e}^{\,j} \text{ or } \frac{t_n-t}{t_n-t_0}\boldsymbol{s}_e^j\,\widehat{\psi}_{e}^{\,j}\,, \tag{6}$$

respectively. We implement geomorphing by simulating a set of concurrent processes, each managing one running geomorph. During each pass through the algorithm, all currently running geomorphs are advanced. When a geomorph is completed, its "process" is removed from the set.

During the adaptation algorithm, as the cut moves up and down, wavelets are added to and removed from the set U_t (Figure 3.4(b)). When a wavelet is added to the set, a new process is created to smoothly add the wavelet to the geometry via a geomorph; similarly, when a wavelet is removed, a new process is created to smoothly remove the wavelet.

Note that, since wavelets can be added and removed independently of each other, geomorphs can overlap in both space and time; that is, a wavelet and some of its descendants may be geomorphing simultaneously. It is also possible for a wavelet to be in the midst of a geomorph when it is added or removed. For example, a wavelet may be added to U_t, causing a geomorph to start, and then it may be removed before the geomorph has completed. In such a case, the removal geomorph will start at the current state of the older geomorph.

Deciding the proper length and speed of geomorph advancement is still an open problem. In our implementation, every geomorph lasts a user-specified number of frames. We have found that geomorphs that last approximately a second give good results. It may be possible to obtain better results by causing the geomorph length to depend on the magnitude of the coefficient of the wavelet being introduced.

The proper geomorph length may also depend on the type of motion being applied to the viewpoint. If the viewpoint is stationary or nearly stationary relative to the object, popping will be noticeable; thus, geomorphs should proceed relatively slowly. The visual effect of changes in the level of detail will be less noticeable when the object is being moved or rotated rapidly relative to the viewer, and faster geomorphs will be possible. Our current implementation includes a crude correction for object motion, scaling geomorph speed linearly with the rotation and translation speed of the viewpoint, using user-specified constants. A more principled approach should understand the perceptual properties of geomorphs in tandem with object motion

In other cases, geomorphs are not necessary at all. If a wavelet is added or removed on the basis of the view frustum or back-facing criteria, the effect of a geomorph will not be visible [10]. We set the geomorph length to zero in those cases.

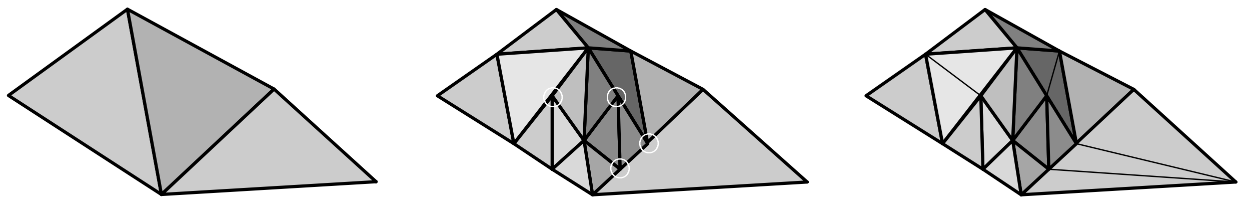

Figure 7 Removing T-vertices. Left: coarse mesh. Center: mesh after adding two wavelets (T-vertices are circled). Right: T-vertices eliminated.

3.6 Rendering

The rendering stage of the algorithm must take the mesh generated for a frame t, and output a series of triangle specifications to pass to the graphics pipeline. The mesh $M_t = \boldsymbol{\rho}_t(K^0)$ is determined by the current subset U_t, plus the state of running geomorphs. Our algorithm iterates over the triangles in M^0. For each triangle considered, it checks whether the triangle should be subdivided to support the current approximation. If so, the algorithm recursively considers each triangle formed by performing a four-to-one subdivision of the triangle. Otherwise, the triangle is drawn with vertex positions set according to $\boldsymbol{\rho}_t$.

Because constructing M_t will typically require non-uniform subdivision, "T-vertices" will often be present (Figure 7). Such vertices will only appear along the edges forming the boundary of the support of a wavelet; thus the actual position of the vertex will be at the midpoint of its edge. But slight cracks may still appear due to roundoff errors. We eliminate those cracks using a recursive triangle cutting algorithm. Before drawing a triangle, we test each edge to see if the triangle across the edge is subdivided. If so, we cut the triangle in half by adding a vertex at the midpoint of the edge, and recursively test each half.

4 Results

We tested our algorithm using three models: a synthetic model of a sphere generated by five recursive four-to-one subdivisions of the octahedron, an acquired model of a small fish statue, and an acquired model of a small elephant statue. The objects were scanned using a laser range scanner, and were constructed triangle mesh models using the volumetric method of [4]. We then parameterized the surfaces using the MAPS algorithm described by Lee *et al.* [11]. The parameterization was then used to generate subdivision connectivity remeshings of each model. The fish was remeshed uniformly by applying four recursive subdivisions. The elephant remesh used a higher error tolerance, and was subdivided three times.

To evaluate the performance of our algorithm, we recorded the movements in an interactive viewing session with each of the three test meshes. In each case, the session included rotating the mesh at a distance in which the entire mesh lay in the view port, followed by zooming in for a closeup view. We then reproduced each session under a variety of settings, recording the number of triangles in the resulting mesh, the number of wavelets in U_t and the time taken to render each frame. The results are illustrated in Figures 8 and 9.

Figure 8 shows performance using a synthetic sphere of 8192 faces generated using five uniform 4-1 subdivisions of an octahedron. The screen-space error tolerance was set such that the error at the silhouette matched the error in the full model. The interior error was set higher by a factor of 7. Figure 8(a) shows the number of faces generated by the adaptation algorithm over time, compared with the total number of faces in the full model. Note that the face count drops off at the end when the camera moves in for a close-up, allowing much of the model to be simplified due to the view frustum test. Figure 8(b) shows the "wall clock" time taken to render each frame, in milliseconds, both for the full model and for the adapted model. The fraction of the LOD rendering time used for the adaptation algorithm is shown as "overhead." Times were recorded on an SGI O2 with a 175 MHz MIPS R10000 with the standard graphics hardware.

Figure 8(c) and (d) show results for an acquired model, a fish statue. The base mesh of 199 triangles was subdivided four times for a final model consisting of 50,944 triangles. Figure 8(c) shows face count, and Figure 8(d) shows rendering times on the SGI O2.

Figure 9 demonstrates the effect of using a spheroid bounding shape versus the shape proposed by [8]. Face counts and render times improve noticeably when viewing at a distance. During closeups, however, the view frustum test takes on greater importance than the screen space error test; consequently, the difference is much less pronounced.

Figure 10 demonstrates the level-of-detail algorithm in action. The fully tessellated sphere of 8192 triangles is decimated to 732 triangles by specifying a silhouette error tolerance of 0.5 pixels and an interior tolerance of 3.5 pixels. In (c), we see the same adapted sphere mesh and the original view frustum from a viewpoint above the original, demonstrating the effect of the three different tests. Regions facing away from the camera or outside the view frustum are significantly coarsened, and a strip of small triangles is clearly visible along the silhouette of the mesh.

Figure 11 shows a scanned elephant model rendered using adaptive refinement (using surface light field rendering [19] instead of simple texture mapping). There are no signs of texture distortion and regions outside of the view frustum are highly simplified. Figure 12 compares an adaptively refined mesh with coarse and fine uniform subdivision. The fish base mesh contains 199 triangles, the adaptive mesh contains 3943 triangles (using a silhouette tolerance of 2 pixels and an interior tolerance of 10 pixels), and the three-times subdivided model contains 12,736 faces. Figures 12 (d), (e) and (f) show the same three meshes with texture applied. Note that the polygonal silhouettes are clearly noticeable in the coarse model although the interior appears plausible. The adapted model and the finely tessellated model appear nearly indistinguishable with texture applied.

5 Summary and future work

We have presented a theoretical framework and a practical implementation of view-dependent level-of-detail for triangle meshes with subdivision connectivity, based on a lazy wavelet analysis. Meshes with this restricted connectivity can be readily generated from surfaces of arbitrary topology using existing surface parameterization algorithms. We implement the same basic capabilities provided with existing progressive mesh-based schemes, including several view-dependent tests, incremental adaptation and runtime generation of geomorphs augmented with more geomorphing features and better bounds on screen-space error. Our method extends the practicality of view-dependent LOD to subdivision connectivity meshes. It also is the first demonstration of adaptive view-dependent LOD for textured meshes of arbitrary topology.

There are several areas for future work. Some features of [8] were not implemented in our system: runtime generation of triangle strips and regulation of the screen-space error tolerance to maintain a constant frame rate. The restricted connectivity of our meshes may make highly efficient generation of triangle strips possible.

Additionally, this work should be extended to interact well with other common applications of subdivision connectivity meshes, including multiresolution mesh editing and animation. The level of detail appropriate for a particular viewpoint includes all of the detail necessary to directly edit the mesh rendered from that viewpoint in a reasonable manner, analogous to multiresolution painting [1].

References

[1] D. F. Berman, J. T. Bartell, and D. H. Salesin. Multiresolution painting and compositing. *Proceedings of SIGGRAPH 94*, pages 85–90, July 1994.

[2] A. Certain, J. Popović, T. DeRose, T. Duchamp, D. Salesin, and W. Stuetzle. Interactive multiresolution surface viewing. In *SIGGRAPH'96*, Computer Graphics Annual Conference Series, pages 91–98, August 1996.

[3] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. *Proceedings of SIGGRAPH 98*, pages 115–122, July 1998.

[4] B. Curless and M. Levoy. A volumetric method for building complex meshes from range images. *Computer Graphics (SIGGRAPH '96 Proceedings)*, 30:303–312, 1996.

[5] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. *Computer Graphics (SIGGRAPH '95 Proceedings)*, 29:173–182, 1995.

[6] T. Funkhouser and C. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 27:247–254, August 1993.

[7] H. Hoppe. Progressive meshes. *Computer Graphics (SIGGRAPH '96 Proceedings)*, 30:99–108, 1996.

[8] H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH'97*, Computer Graphics Annual Conference Series, pages 189–198, August 1997.

[9] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *Proceedings of SIGGRAPH 93*, pages 19–26, August 1993.

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

[11] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98*, pages 95–104, July 1998.

[12] M. Lounsbery, T. DeRose, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, January 1997.

[13] M. Lounsbery, T. D. DeRose, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, January 1997.

[14] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH'97*, Computer Graphics Annual Conference Series, pages 199–208, August 1997.

[15] J. Rossignac and P. Borel. Multi-resolution 3d approximations for rendering. In *Modeling in Computer Graphics*, pages 455–465, June 1993.

[16] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):65–70, July 1992.

[17] E. Stollnitz, T. DeRose, and D. Salesin. *Wavelets for Computer Graphics: Theory and Applicaitons*. Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers, Inc, San Francisco, 1996.

[18] G. Turk. Re-tiling polygonal surfaces. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):55–64, July 1992.

[19] D. N. Wood, D. I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. Salesin, and W. Stuetzle. Surface light fields for 3d photography. *Proceedings of SIGGRAPH 2000*, pages 287–296, July 2000.

[20] J. C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. *IEEE Visualization '96*, pages 327–334, October 1996.

[21] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. *Proceedings of SIGGRAPH 97*, pages 259–268, August 1997.

A Derivation of error

We want the screen projection of the parametric error to be less than τ. This criterion can be written

$$\max_{p\in\operatorname{supp}\left(\widehat{\psi}_e{}^j\right)} E_{scrn}(p) \approx \frac{\left|\boldsymbol{E}_{param}(p)\times \frac{\boldsymbol{v}_e^j-\boldsymbol{q}_t}{|\boldsymbol{v}_e^j-\boldsymbol{q}_t|}\right|}{2\cot\left(\frac{\varphi}{2}\right)} < \tau\,. \tag{7}$$

with $\boldsymbol{q}_t$ the camera center, and for all p in the support of $\widehat{\psi}_e{}^j$, provided that both the support of $\widehat{\psi}_e{}^j$ is small and the parametric error is small relative to the distance to the camera. Or, formally:

$$\text{(i)}\quad \frac{\boldsymbol{\rho}(p)-\boldsymbol{q}_t}{|\boldsymbol{\rho}(p)-\boldsymbol{q}_t|} \approx \frac{\boldsymbol{v}_e^j-\boldsymbol{q}_t}{|\boldsymbol{v}_e^j-\boldsymbol{q}_t|}$$

$$\text{(ii)}\quad \frac{\boldsymbol{\rho}(p)-\boldsymbol{\rho}_t(p)}{\boldsymbol{v}_e^j-\boldsymbol{q}_t} \text{ is small.}$$

The projection into the viewing plane of the bounding ellipsiod is an ellipse. A messy, but straight forward (in Mathematica), computation shows that the formulas for squared lengths of the semimajor and semiminor axes are

$$\frac{{a_e^j}^2(|\boldsymbol{v}_e^j-\boldsymbol{q}_t|^2-{b_e^j}^2)-({a_e^j}^2-{b_e^j}^2)\left(\hat{\mathbf{n}}\cdot(\boldsymbol{v}_e^j-\boldsymbol{q}_t)\right)^2}{\left\{|\boldsymbol{v}_e^j-\boldsymbol{q}_t|^2-{b_e^j}^2-({a_e^j}^2-{b_e^j}^2)\left(\hat{\mathbf{n}}\cdot\frac{(\boldsymbol{v}_e^j-\boldsymbol{q}_t)}{|\boldsymbol{v}_e^j-\boldsymbol{q}_t|}\right)^2\right\}^2}$$

and

$$\frac{{b_e^j}^2}{|\boldsymbol{v}_e^j-\boldsymbol{q}_t|^2-{b_e^j}^2-\left({a_e^j}^2-{b_e^j}^2\right)\left(\hat{\mathbf{n}}\cdot\frac{(\boldsymbol{v}_e^j-\boldsymbol{q}_t)}{|\boldsymbol{v}_e^j-\boldsymbol{q}_t|}\right)^2}\,,$$

respectively.

For $(\boldsymbol{v}_e^j-\boldsymbol{q}_t)$ large relative to the error, we may assume orthographic projection, in which case, these formulas reduce to the following:

$$\frac{{a_e^j}^2|\boldsymbol{v}_e^j-\boldsymbol{q}_t|^2-({a_e^j}^2-{b_e^j}^2)\left(\hat{\mathbf{n}}\cdot(\boldsymbol{v}_e^j-\boldsymbol{q}_t)\right)^2}{|\boldsymbol{v}_e^j-\boldsymbol{q}_t|^4}$$

and

$$\frac{{b_e^j}^2}{|\boldsymbol{v}_e^j-\boldsymbol{q}_t|^2}\,.$$

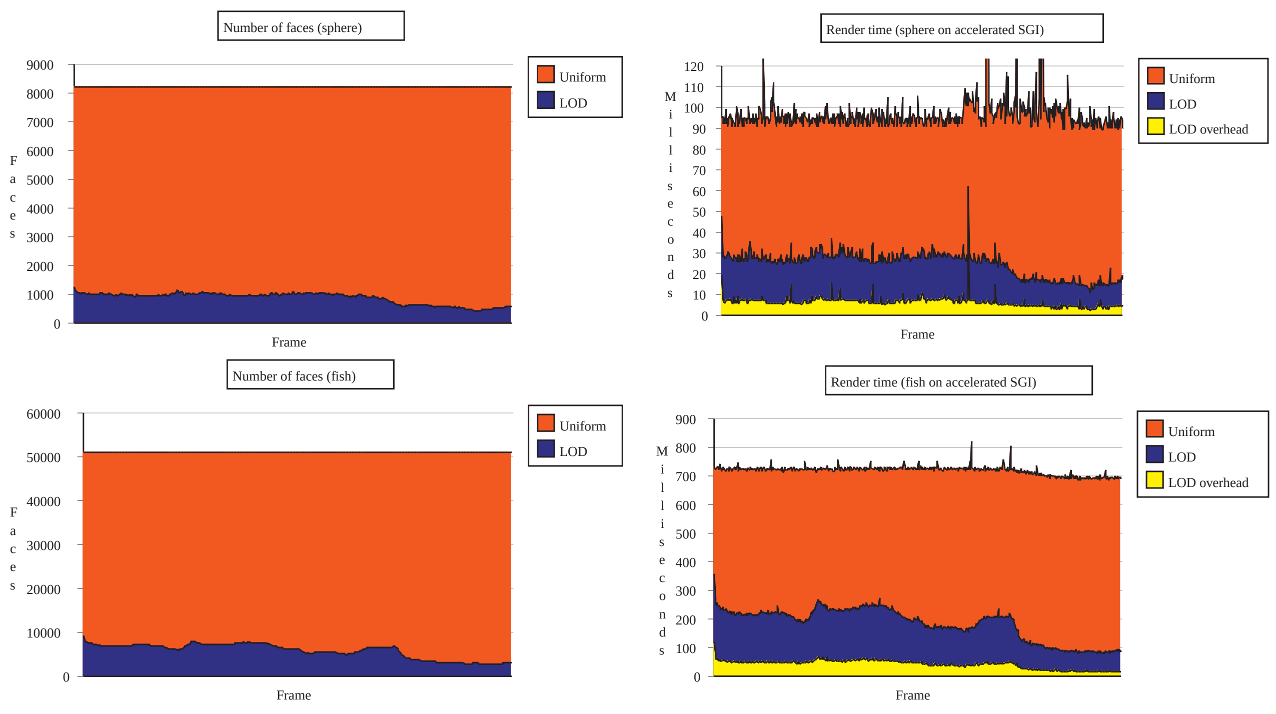

Figure 8 (a) Face count for sphere model (left) (b) Rendering time for sphere model (right). (c) Face count for fish model. (top-left) (d) Rendering times for fish model on SGI. (top-right).

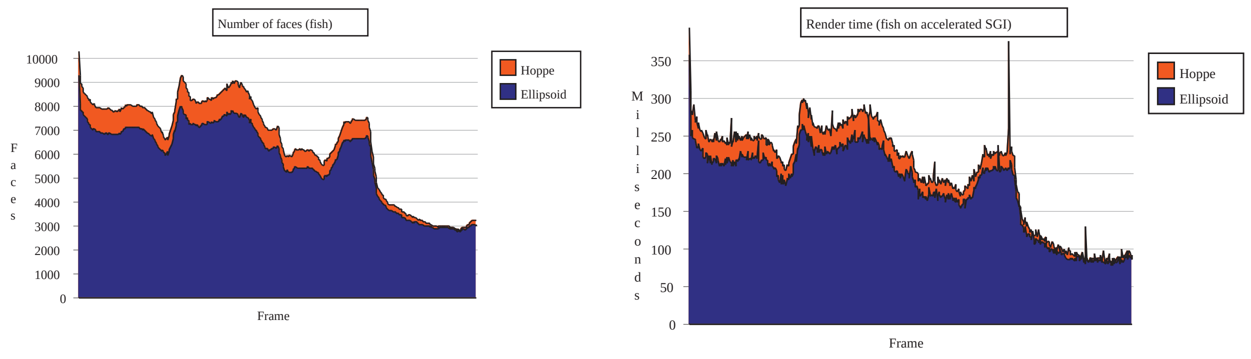

Figure 9 (a) Face count for different bounding shapes (b) Rendering times for different bounding shapes.

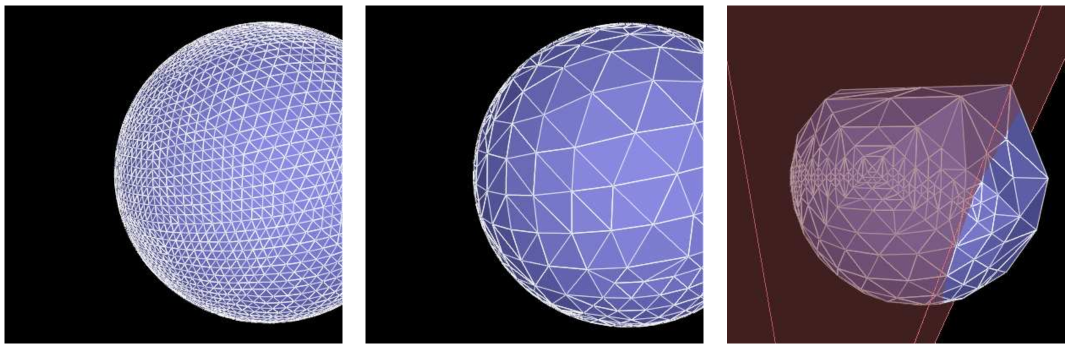

Figure 10 From left to right: (a) Full sphere model (8192 faces). (b) Adapted sphere (732 faces). (c) Adapted sphere and view frustum, shown from alternate viewpoint.

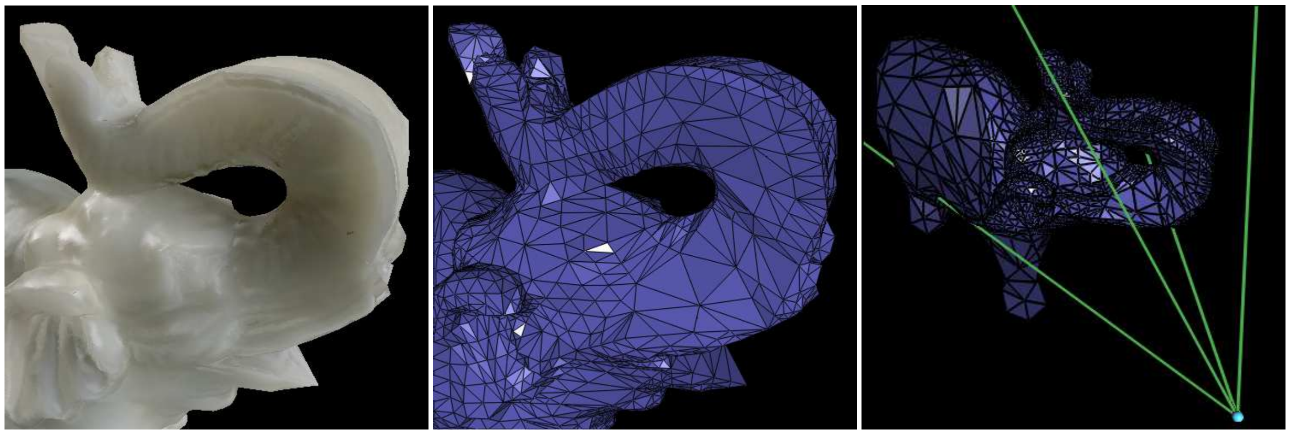

Figure 11 From left to right: adaptive refinement of elephant model rendered with flat-shaded triangles, adaptive refinement rendered using surface light fields [19], and the adaptive refinement shown from alternate viewpoint.

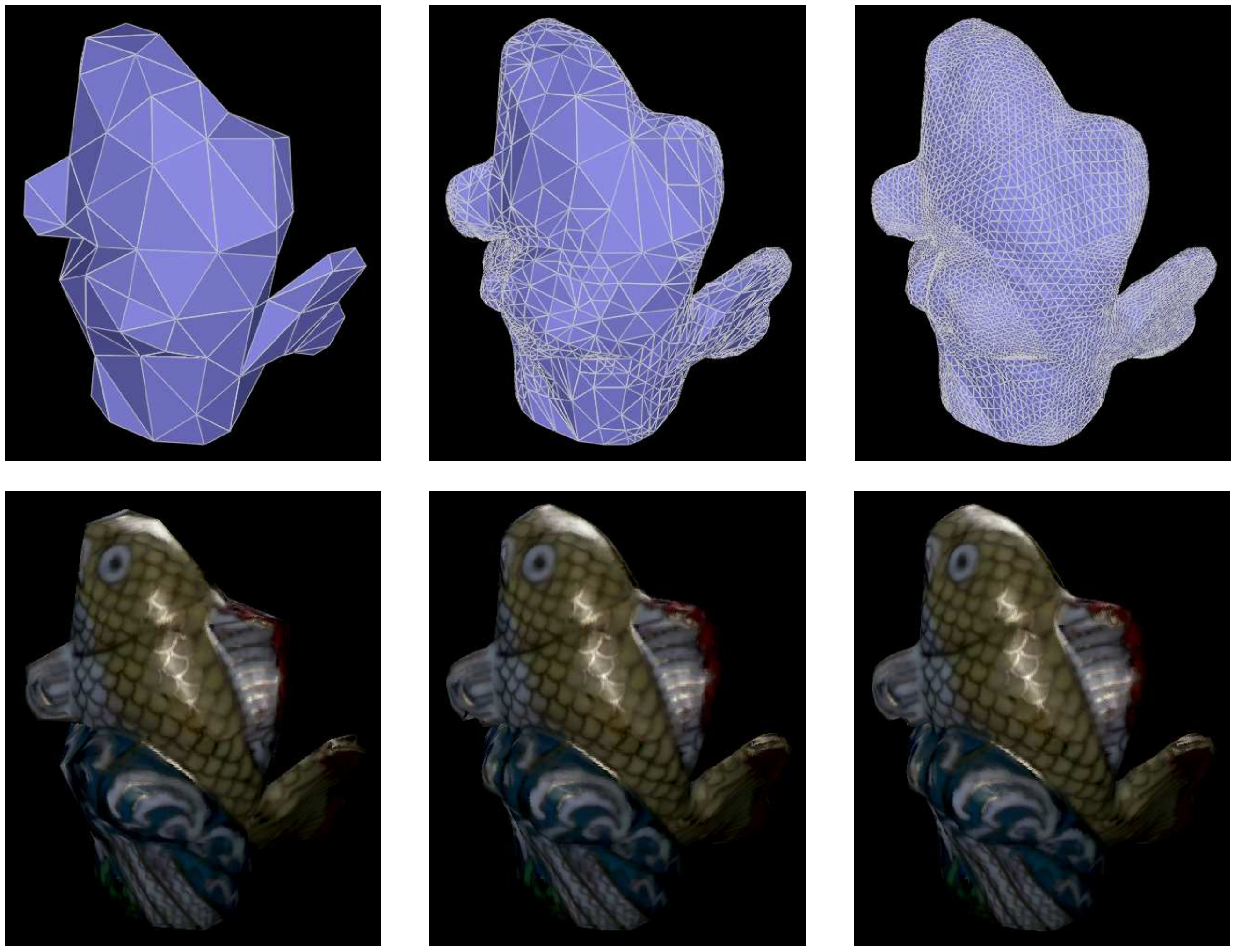

Figure 12 From left to right, top to bottom: (a) Coarse geometry (199 faces). (b) Adapted geometry (3943 faces). (c) Fine geometry (12,736 faces). (d) Coarse textured geometry. (e) Adapted textured geometry. (f) Fine textured geometry.