To appear in ACM Transactions on Graphics

Image-Driven Simplification

Peter Lindstrom Greg Turk

Georgia Institute of Technology

Abstract

We introduce the notion of *image-driven simplification*, a framework that uses images to decide which portions of a model to simplify. This is a departure from approaches that make polygonal simplification decisions based on geometry. As with many methods, we use the edge collapse operator to make incremental changes to a model. Unique to our approach, however, is the use of comparisons between images of the original model against those of a simplified model to determine the cost of an edge collapse. We use common graphics rendering hardware to accelerate the creation of the required images.

As expected, this method produces models that are close to the original model according to image differences. Perhaps more surprising, however, is that the method yields models that have high geometric fidelity as well. Our approach also solves the quandary of how to weight the geometric distance versus appearance properties such as normals, color and texture. All of these tradeoffs are balanced by the image metric. Benefits of this approach include high fidelity silhouettes, extreme simplification of hidden portions of a model, attention to shading interpolation effects, and simplification that is sensitive to the content of a texture. In order to better preserve the appearance of textured models, we introduce a novel technique for assigning texture coordinates to the new vertices of the mesh. This method is based on a geometric heuristic that can be integrated with any edge collapse algorithm to produce high quality textured surfaces.

Keywords: Polygonal simplification, level-of-detail, image metrics, visual perception.

1 INTRODUCTION

By far the most common use of automatic polygonal simplification is to produce a model that is visually similar to the original model yet contains fewer polygons. The key phrase is *visually similar*. Although the goal of simplification is visual similarity, most methods concentrate on a related yet distinct goal, geometric fidelity. Every published method that we know uses some measure of 3D distance (occasionally using other factors as well) to guide the simplification. This is quite reasonable since much of the appearance of an object is dictated by geometry. In this paper, however, we present a simplification method that is guided by visual similarity, as measured by image differences rather than 3D distances. We take this approach to better match the goal of producing models that appear similar to a human observer.

Purely geometric simplification methods use the positions of vertices, edges and faces to decide how close a simplified model is to the original. Recently a few new simplification methods have also incorporated surface properties into their measure of deviation from the original model. Such properties include color, surface normals and texture coordinates. The most common way in which to incorporate surface properties is to modify the distance measure (that begins as purely 3D distance) by adding a weighted sum of deviations for each of the surface properties. How should the weights in such an approach be chosen? Is there a way in which to decide how these different properties interact with one another? By creating

images of the original and the simplified models, we can use a measure of visual distance to evaluate the closeness of the models. Interactions between the various properties is handled in a unified fashion by the single image metric. Furthermore, as we learn more about human visual perception, this approach can be enhanced by using new perceptually-based image metrics.

Before describing our method in more detail, it is worth considering which applications can make use of such a simplification method. A small number of applications require exact geometric tolerances with respect to an original model. Such applications include collision detection and path planning for part insertion and removal. In these areas, geometric distance to the original model is necessary. More common, however, are applications in which the requirement is for the simplified model to be visually similar to the original model. Examples of such applications include vehicle simulations, building walk-throughs, video games, acceleration of off-line rendering, and fast manipulation of simplified models that can then be rendered at high fidelity after the user has chosen a suitable view. In these applications, geometric proximity takes a back seat to visual fidelity, and an image-driven simplification method is well-suited to the tasks.

The remainder of this paper is organized as follows. In the next section we review work in polygonal simplification. Section 3 describes a simple image metric and demonstrates how it may be used as a quality measure of a simplified model. Section 4 demonstrates the use of this quality measure for driving an edge collapse simplification algorithm. It describes a hardware implementation of these ideas that uses the OpenGL graphics API and common graphics hardware. In Section 5 we analyze the performance of the image-driven simplification method. Section 6 concludes with a discussion of future work.

2 PREVIOUS WORK

A common theme in simplification is the use of either an explicit or an implied geometric distance measure between the original and the simplified model to guide the simplification process. We will highlight these distance measures in our overview of previous work. Because of the large number of published articles on simplification, our review will necessarily be incomplete.

Several simplification approaches rely on successive vertex removal and local re-triangulation to create models with fewer geometric primitives. Schroeder and co-workers use such a vertex removal method that is guided by how far a given vertex is to a plane that is fit to neighboring points of the candidate vertex [34]. This approach has more recently been augmented to allow changes to the topology of the surface [35]. Ciampalini and co-workers enhanced the decimation approach by maintaining two estimates of the global errors introduced by removing vertices [3]. They further improved the method by allowing edge flips during simplification to better match the original surface. To guarantee a given distance tolerance ϵ between the original and simplified model, Cohen and co-workers build an internal and an external envelope around the given model that avoids self-intersection and that is no further than ϵ from the original surface [5]. Each vertex is tested by attempting to remove it and re-triangulate the hole in such a manner that the resulting surface does not intersect either envelope. Rather than removing one vertex at a time, Rossignac and Borrel cluster all the vertices of a model into the cells of a grid of cubes and then replace all of the vertices that fall into a single cell by a single representative vertex [32]. In this manner, they too are able to maintain an upper bound ϵ on the Hausdorff distance between the two models, where ϵ is the length of the diagonal of a cell.

Another local operation used for simplification is edge collapse: two vertices that share an edge are removed and replaced by a single vertex. Hoppe and co-workers use edge collapse, edge swap, and edge split operations to simplify models [17]. These operations are guided by an energy function that measures

the distance from the current simplified model to a set of points in 3D that sample the original model. In more recent work, Hoppe relies entirely on edge collapse operations [18]. This work is notable in that it is one of the earliest attempts to preserve surface color as well as geometry. This is accomplished by including a term for color in the energy function, as well as terms for preserving sharp features and material boundaries.

Ronfard and Rossignac, too, use successive edge collapse operations that are guided by a distance measure. Each vertex of the original model has associated with it a list of the planes of the adjacent faces to the vertex [31]. The position of the new vertex from a potential edge collapse is chosen to be one of the two original vertices of that edge. The cost of performing the edge collapse is determined by the maximum distance to the set of planes associated with the edge's vertices, and a priority queue is used to order the edge collapse operations. The vertex associated with an edge collapse acquires the lists of planes from both vertices of the edge. Inspired by this technique, Garland and Heckbert convert a set of implicit plane equations into a symmetric 4 by 4 matrix that allows the rapid computation of the sum of the square of the distances between a given point and the planes [13]. They use this new distance measure to guide a series of vertex pair contractions, a generalization of edge collapse that allows topological changes to a model. Kobbelt et al. use edge collapse for their simplification method that is guided by a weighted measure of the Hausdorff distance, the local distortion, and the local curvature [21].

Most of the simplification methods that have been published produce static models from an off-line simplification algorithm. A few researchers have built a sequence of progressively simplified meshes and have then used a run-time algorithm to selectively refine the model based on the observer's viewpoint. Xia and Varshney demonstrated such a run-time system that selected the triangles to display based on their screen-space area, whether they were on the silhouette, and if there was a specular highlight nearby [37, 38]. More recently, El-Sana and Varshney have enhanced this work using a view-dependence tree to help avoid foldover and for better memory utilization [9]. The same researchers have also described a method of producing triangle strips on-the-fly for view-dependent simplification [10]. Hoppe performed a careful analysis of edge-collapse dependencies to create a view-dependent algorithm for displaying surfaces [19]. His approach exploits frame coherence and moves vertex positions over several frames to eliminate popping artifacts. Luebke and Erikson perform view-dependent simplification based on a hierarchical data structure that can be built off-line using vertex clustering, envelopes or progressive meshes [27]. Gu et al. use an off-line construction process to create a number of silhouettes, one of which is selected at run-time to clip the silhouette of a simplified, texture-mapped model in order to avoid noticeable faceting [16]. All of the above view-dependent methods use geometric distance measures to perform their off-line simplification preprocessing. In particular, the preprocessing is typically done in a *view-independent* manner, and it is the responsibility of the *run-time* system to construct a view-dependent model by choosing from a set of predefined simplification moves (e.g. edge collapse, vertex removal, etc.) which, if not constructed carefully, are not likely to optimally preserve model appearance. Because our image-driven simplification method accounts for model appearance *during* the off-line preprocessing stage, we believe that the progressive meshes it produces could be used more effectively in a view-dependent run-time system than those obtained via geometry-driven methods.

Recently there have been a few published methods that take into account surface properties during the simplification process. Garland and Heckbert have augmented their error metric to include terms for surface properties such as texture coordinates and color [14]. Hoppe recently improved upon their work by proposing an alternative, more compact quadric error metric for measuring differences in geometry and surface attributes, and found that the use of "memoryless" edge collapses (see [24]) further improved model quality [20]. Erikson and Manocha use point clouds in color space and texture coordinate space to choose new material properties after performing a vertex merge [11]. Cohen and co-workers describe an

edge collapse algorithm that maintains a mapping between the original and simplified surface [6]. This allows them to maintain error bounds for texture coordinates. Their technique was later extended to emphasize appearance preservation by parameterizing the surface and encapsulating surface attributes and fine details in normal and texture maps [7]. While motivated by the same goal, their approach is quite different from ours in that they rely on converting the model to an alternate representation. In order to display such an augmented model, specialized hardware or software algorithms are needed. For textured models, previous simplification methods concern themselves only with the coordinates of the texture parameterization. In contrast to this, our own method of simplifying models with texture is sensitive to the color variations within the texture.

3 COMPARING MODELS USING IMAGES

In this section we lay the groundwork for using comparisons between images to steer the simplification of a model. First, we outline a simple per-pixel-difference image metric that is fast yet quite effective at capturing the visual aspects that are important for simplification. Next, we describe how multiple images of the original and a simplified model (taken from a sphere of camera positions) can be compared in order to judge the similarity of the models. Later in the paper this form of comparison will provide the cost for a proposed edge collapse during simplification. The final sub-section discusses the similarity between our sphere of images and other collections of images that have been used for image-based rendering.

3.1 A Simple Image Metric

An *image metric* is a function over pairs of images that gives a non-negative measure of the distance between the two images. Many image metrics have been proposed recently, of which several attempt to capture some of the fundamental aspects of human visual perception. The point is often made that a metric based on pixel-wise differences—such as the root mean square (RMS) error—is not a good measure of visual similarity, and pathological examples of its inadequacy are often demonstrated. Such examples include image distortion (e.g. random or oriented noise, superimposed sinusoid gratings, compression artifacts, etc.) and affine transformations (e.g. translations, rotations, and scaling of the image contents). There are, however, applications for which the simple RMS measure is a fair indicator of visual similarity, and we use this metric because we have found it to be well-suited to our task. In all but a very few cases, image distortion and transformations don't arise in our application. As a potential problem, one might suspect that the RMS metric would overemphasize image differences due to minor shifts in the texture parameterization of a model, particularly if the texture is rich in high frequency content. However, as shown in Section 5, using the RMS metric to guide simplification leads to results that are substantial improvements over those obtained without an image metric. The RMS metric also has the virtue of being fast to compute, which is important for guiding simplification. We note that our image-driven simplification framework easily allows other image metrics to be used, and we see this as a fruitful area for future work.

Even though some of our examples include colored models, we compute a single luminance channel Y for each image and measure only differences in luminance, which has worked well for all of our test models. Of course, there is nothing inherent in our approach that precludes us from taking differences in hue into account, yet such a metric would require additional storage and computation, and would add little other than the ability to distinguish variations in color between equiluminant regions; a feature that we haven't found to be of any greater significance. To compute Y, we use the standard NTSC coefficients to

weight the RGB triplets. Formally, we define the RMS difference between two luminance images Y^0 and Y^1 of dimensions $m \times n$ pixels as

$$d_{RMS}(Y^0, Y^1) = \sqrt{\frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(y_{ij}^0 - y_{ij}^1\right)^2} \tag{1}$$

3.2 Comparing Models Using Multiple Views

We cannot hope to capture the entire appearance of an object in a single image. Ideally, we wish to capture the set of all radiance samples that emanate from the surface of an object under all possible lighting conditions. This is obviously not possible in practice. To capture a large collection of radiance samples of the object, we render images from a number of different camera positions around the object and apply the image metric to this entire set of images. While the number of views required and the "optimal" placement of viewpoints vary between objects, we have chosen to fix the set of viewpoints, and require that they are distributed uniformly in direction. For the models included in this paper, a fixed set of camera positions has proven adequate, even for the less uniformly shaped objects, although it is possible that these viewpoints do not yield optimal or even total coverage of more complex surfaces. Promising methods are currently being developed to address this optimization problem [36]. For each image of a model, we use a single light source, fixed near the viewer, such that the front of the model (as seen from each viewpoint) is always illuminated.

To ensure even coverage of image samples, we arrange our viewpoints so as to approximate a sphere of camera positions surrounding an object. We put the set of images that we generate to two distinct uses: 1) determination of the cost of an edge collapse, and 2) evaluation of the visual quality of a model, regardless of whether it was created based on geometric or visual criteria. We will return to this second use in Section 5 when we compare image-driven simplification to other simplification methods. For determining edge costs, we place the cameras at the vertices of a regular dodecahedron, which gives 20 images. For quality evaluation of a model, we use the 24 vertices of the *small rhombicuboctahedron*. We deliberately use a different set of camera positions for these two cases to avoid too much bias towards models that were created using an image-driven method when we compare methods. Also to avoid bias, we use PhotoRealistic RenderMan to produce high-quality 512×512 images for quality evaluation, but we use hardware rendering to generate 256×256 images for evaluating edge costs. In both cases, we place the model on a gray (50% intensity) background. Even though it is possible for parts of the surface silhouette to blend into the background, leading to an apparent loss in visual importance, it is extremely unlikely that the same surface patch would be "camouflaged" in several views. In addition, silhouettes often approach tangency to the view direction, leading to a sharp intensity gradient and causing the silhouettes to appear dark as relatively little light is reflected back to the viewer.

To ensure that the model is entirely contained in each image, we compute the minimum bounding sphere of its vertices, double the radius of this sphere, and constrain the viewpoints to lie on the surface of this larger sphere. The camera at each viewpoint is then directed towards the center of the sphere, and the camera up-vectors are varied uniformly. We use a 60° field of view perspective projection such that the projection of the bounding sphere is exactly contained in each square image. When comparing images, we use the same camera parameters to render both objects. Figure 3 shows a uniform distribution of views of the Stanford bunny model.

Given two sets of l images $\mathcal{Y}^0 = \{Y_h^0\}$ and $\mathcal{Y}^1 = \{Y_h^1\}$ of dimensions $m \times n$ pixels, we compute the

RMS difference as

$$d_{RMS}(\mathcal{Y}^0, \mathcal{Y}^1) = \sqrt{\frac{1}{lmn} \sum_{h=1}^{l} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(y^0_{hij} - y^1_{hij}\right)^2} \tag{2}$$

That is, we treat the set of images as a single image mosaic and assume that the domain is some arbitrary set of independent pixel pairs. The RMS difference satisfies all the mathematical properties of a metric. We will see later how using the mean square (MS) difference can be used to simplify the evaluation of the metric. While MS does not satisfy the triangle inequality of a metric, it has a one-to-one mapping with RMS. In general, we are not concerned with the actual magnitude of the RMS error, but rather focus on using it to order sets of images by increasing mean square difference.

3.3 Comparison to Light Fields

The rendering community has recently devoted attention to the idea of capturing a large portion of the *light field* or *plenoptic function* of a given scene. A light field is a 5D function that represents the radiance at a given 3D point in a given direction. Several groups of researchers have used sampled representations of a light field for image-based rendering [15, 22, 29]. Recognizing that in free space the radiance does not change along a ray, the dimension of the light field around an object can be reduced to 4D. One such representation of the light field, used by [15, 22], is a collection of images taken from a 2D array of camera positions on a flat surface. Our own sphere of camera positions that we use for comparing models is also a 4D representation of the radiance of an object. Two of the dimensions are from pixel position in an image and two are from the camera position on the sphere surrounding the object. At the limit, an infinitely dense set of camera positions on the sphere and infinite image resolution would give a complete representation of the free-space light field for an object. This assumes that the camera viewing direction and the field of view are set so that no part of the model lies outside the image, as is the case for our images.

One difference between our sphere of images and the collections of images that are used for image-based rendering is the lighting of the object. Most image-based rendering techniques assume fixed lighting for a given object or scene. We choose to position the light differently for each camera position, fixed to the *viewer*, in order to bring out a large amount of detail in each image. The space of radiance samples under all possible lighting conditions is an infinite dimensional space, therefore compromises must be made in order to sample this space using finite computational resources and memory.

4 IMAGE-DRIVEN SIMPLIFICATION

In this section, we present an image-driven simplification algorithm which makes use of the popular edge collapse operation (Figure 1). This operation merges two vertices that share an edge and replaces them with a new vertex, while removing the triangles bounded by the edge. The general edge collapse algorithm is greedy, i.e. the edges are ordered in a priority queue and the "cheapest" edge is collapsed in each iteration. Given such an edge collapse sequence, the model is simplified until a desired number of triangles remain, or until some error threshold is exceeded.

As with any edge collapse algorithm, two fundamental decisions must be made: 1) where to place the new vertex, and 2) how to order the edge collapses. Typically, the latter involves minimization of some objective function, or *edge cost*, which depends on the position of the replacement vertex. In our algorithm, we use a heuristic for the vertex placement that is entirely geometry-based, while the edge cost function is given by an image metric. In the following subsections, we describe our strategies for vertex

placement and edge ordering. We will make frequent use of the two simplex operators $\lfloor s \rfloor$ (the *faces* or $n-1$-simplices that make up an n-simplex s) and $\lceil s \rceil$ (the $n+1$-simplices that s is a subset of) [24]. For a manifold edge e, $\lfloor e \rfloor$ gives its two vertices, while $\lceil e \rceil$ denotes the two triangles that share e. The extension to sets is trivial, e.g. $\lceil \lfloor e \rfloor \rceil = \bigcup_{v \in \lfloor e \rfloor} \lceil v \rceil$. Figure 2 illustrates these simplex operators. In the remainder of this section, we describe vertex placement, texture coordinate selection, and finally our new image-based edge cost.

4.1 Memoryless Vertex Placement

Vertex placement is implicitly an optimization problem; given an edge e to collapse, we wish to choose the position of the new vertex v such that the associated objective function (edge cost) is minimized. In our algorithm, the objective function is expressed in terms of rendered images of an object, and cannot as such be written as a simple function of v. Several global optimization methods exist for solving such problems, although these generally require a large number of evaluations of the objective function to find the optimum. Rather than undertaking such a difficult task, we use the simple heuristic introduced in [24], which has proven successful both in conjunction with our image-based edge cost and in our original geometry-based scheme. One of the benefits of this vertex placement scheme is that it can be used with any edge collapse algorithm, and does not require maintaining any additional information about the original model or the history of edge collapses; it is entirely "memoryless" and uses only information about the model in the current iteration of the simplification. Actually, any vertex placement method may be used in conjunction with our image-based edge cost. We use memoryless vertex placement because it makes reasonable placement choices and because it is fast and memory efficient. We will briefly describe the basic components of the memoryless algorithm for collapsing non-boundary edges. Boundary edges are treated in a manner analogous to interior edges, but involve computations in two rather than three dimensions. For equations and further details, see [24, 25].

Volume preservation is one of the key features of the memoryless vertex placement. As an edge is collapsed, tetrahedral volumes are swept out (Figure 1). Such volumetric changes are either positive or negative. By setting the sum of volume changes to zero and solving for the vertex position, a plane equation is obtained. Thus, the volume bounded by the local surface patch is preserved by placing the substitute vertex somewhere in this plane.

To further preserve the geometric shape of the object, the replacement vertex is chosen such that the sum of per-triangle squared changes in volume is minimized. In non-planar regions, this fully constrains the vertex. If the surface is locally planar, we choose the vertex such that the resulting triangles are as near equilateral as possible, thereby reducing the probability of introducing unwanted sliver triangles.

All of the steps involved in this vertex placement scheme are analytical and linear in nature, and can be implemented using simple linear algebra operations. In contrast to most contemporary edge collapse methods, the time required to compute the vertex position is a tiny fraction of the time needed to evaluate our edge cost. Thus, we will dedicate the larger part of this paper to discussing the edge cost and how to evaluate it efficiently.

4.1.1 Assignment of Texture Coordinates

For textured models, the new vertex needs to be supplied with texture coordinates after an edge has been collapsed. Several methods have been proposed for this task, some which minimize the resulting distortion of the texture parameterization [6, 11, 14, 20, 23]. Rather than adopting one of these algorithms, we have developed a new method that is based upon and generalizes our selection of vertex coordinates. Similar in

spirit to the work of Garland and Heckbert [14], we extend the representation of vertices from $\mathbb{R}^3$ to $\mathbb{R}^5$, letting the additional two (orthogonal) coordinate axes represent texture space. Similar to [14] and [20], we ensure that our space is scale-invariant, that is, we apply a linear, uniform scaling κ to the texture coordinates to make them "compatible" with the geometric coordinates. However, instead of scaling the model geometry to the unit cube, our scale factor is computed as the ratio of the average geometric edge length to the average texture space edge length. For models with multiple textures (or multiple parameterizations), we compute a different κ for each texture patch.

We have chosen to treat the texture coordinate computation as a black box that takes as input the position $\mathbf{x} = \begin{pmatrix} x & y & z \end{pmatrix}^{\mathrm{T}}$ of the replacement vertex v and the geometry and texture coordinates of the triangles $\{t_i\} = \lceil\lceil\lfloor e \rfloor\rceil\rceil$ surrounding the collapsed edge, and which outputs the texture coordinates $\begin{pmatrix} s & t \end{pmatrix}^{\mathrm{T}}$ for v. Thus, in contrast to [14, 20], the position of v is not influenced by the texture coordinates. Rather, we choose the "best" set of texture coordinates for a given vertex position.

Our goal is to formulate a method that generalizes the volume-preserving and per-triangle volume-minimizing properties [24] of the vertex placement to five dimensions. Consequently, we measure the tetrahedral volumes—embedded in a higher dimensional space, $\mathbb{R}^5$—swept out by the triangles $\{t_i\}$. In [24], we encountered a similar issue when measuring the changes in area described by 2-simplices (triangles) embedded in $\mathbb{R}^3$. Specifically, we developed a method for preserving the shape of boundary loops by minimizing the triangular areas swept out by boundary edges. The additional degree of freedom given by this subspace embedding is used to encode not the sign of the swept out area, but its *orientation*. This orientation is given by a vector $\mathbf{n}$ perpendicular to the swept out triangle with magnitude equal to its area:

$$\mathbf{n} = \frac{1}{2}\begin{vmatrix} \hat{\mathbf{e}}_1 & x_1 & x_2 & x_3 \\ \hat{\mathbf{e}}_2 & y_1 & y_2 & y_3 \\ \hat{\mathbf{e}}_3 & z_1 & z_2 & z_3 \\ \mathbf{0} & 1 & 1 & 1 \end{vmatrix} = \frac{1}{2}\left(\begin{vmatrix} y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \\ 1 & 1 & 1 \end{vmatrix} \begin{vmatrix} z_1 & z_2 & z_3 \\ x_1 & x_2 & x_3 \\ 1 & 1 & 1 \end{vmatrix} \begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix} \right)^{\mathrm{T}} \tag{3}$$

where $\{\hat{\mathbf{e}}_j\}$ are the standard unit basis vectors. Analogous to the oriented area of Equation 3, we can use an *oriented volume* to aid in texture coordinate selection.

For a 3-simplex (tetrahedron) in $\mathbb{R}^5$, we have two additional degrees of freedom, resulting in an entire plane of vectors orthogonal to the simplex, which, with a little abuse of notation, can be encapsulated in a rank 2 skew-symmetric matrix $\mathbf{V}$:

$$\mathbf{V} = \frac{1}{6}\begin{vmatrix} \hat{\mathbf{e}}_1 & \hat{\mathbf{e}}_1^{\mathrm{T}} & x & x_1 & x_2 & x_3 \\ \hat{\mathbf{e}}_2 & \hat{\mathbf{e}}_2^{\mathrm{T}} & y & y_1 & y_2 & y_3 \\ \hat{\mathbf{e}}_3 & \hat{\mathbf{e}}_3^{\mathrm{T}} & z & z_1 & z_2 & z_3 \\ \hat{\mathbf{e}}_4 & \hat{\mathbf{e}}_4^{\mathrm{T}} & \kappa s & \kappa s_1 & \kappa s_2 & \kappa s_3 \\ \hat{\mathbf{e}}_5 & \hat{\mathbf{e}}_5^{\mathrm{T}} & \kappa t & \kappa t_1 & \kappa t_2 & \kappa t_3 \\ \mathbf{0} & \mathbf{0}^{\mathrm{T}} & 1 & 1 & 1 & 1 \end{vmatrix} = \frac{1}{6}\begin{pmatrix} 0 & g_1 & g_2 & g_3 & g_4 \\ -g_1 & 0 & g_5 & g_6 & g_7 \\ -g_2 & -g_5 & 0 & g_8 & g_9 \\ -g_3 & -g_6 & -g_8 & 0 & g_{10} \\ -g_4 & -g_7 & -g_9 & -g_{10} & 0 \end{pmatrix} \tag{4}$$

We consider each product $\hat{\mathbf{e}}_j\hat{\mathbf{e}}_k^{\mathrm{T}}$ of elements in the first two columns of the determinant to be an outer product 5×5 matrix with a single non-zero entry $\hat{e}_{jk} = 1$. Thus, we have for example

$$g_1 = \begin{vmatrix} z & z_1 & z_2 & z_3 \\ \kappa s & \kappa s_1 & \kappa s_2 & \kappa s_3 \\ \kappa t & \kappa t_1 & \kappa t_2 & \kappa t_3 \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

which is linear in the variables s and t. We refer to the matrix $\mathbf{V}$ as the *oriented volume* of a tetrahedron. As a result of our construction, the tetrahedron lies in the 3-dimensional subspace of $\mathbb{R}^5$ that is the null

space of $\mathbf{V}$, analogous to how the 2-dimensional subspace formed by a triangle plane in $\mathbb{R}^3$ is the null space of its normal $\mathbf{n}^{\mathrm{T}}$. The unsigned tetrahedral volume is then $V = \frac{1}{\sqrt{2}}\|\mathbf{V}\|_F$, where $\|\cdot\|_F$ is the Frobenius norm (the square root of the sum of squared matrix elements). Since $\mathbf{V}$ is skew-symmetric, we need only include the 10 elements $\{g_1, \ldots, g_{10}\}$ in its upper triangle when computing the Frobenius norm, and we can factor out the only two unknowns s and t (i.e. the texture coordinates of v):

$$V_i^2 = \frac{1}{2}\|\mathbf{V}_i\|_F^2 = \frac{1}{36}\begin{pmatrix} s & t & 1 \end{pmatrix} \bar{\mathbf{G}}_{T_i}^{\mathrm{T}} \bar{\mathbf{G}}_{T_i} \begin{pmatrix} s \\ t \\ 1 \end{pmatrix} \tag{5}$$

where $\bar{\mathbf{G}}_{T_i}$ is a 10×3 matrix associated with triangle t_i, whose rows correspond to a factorization of the 10 linear expressions of s and t given by $\{g_j\}$. We then use $\bar{\mathbf{G}}_{T_i}$ to define two objective functions f_{T_p} and f_{T_o} for volume preservation and optimization, respectively:

$$f_{T_p}(s,t) = \frac{1}{2}\begin{pmatrix} s & t & 1 \end{pmatrix} \left(\frac{1}{18} \sum_i \bar{\mathbf{G}}_{T_i}^{\mathrm{T}} \sum_i \bar{\mathbf{G}}_{T_i} \right) \begin{pmatrix} s \\ t \\ 1 \end{pmatrix} \tag{6}$$

$$f_{T_o}(s,t) = \frac{1}{2}\begin{pmatrix} s & t & 1 \end{pmatrix} \left(\frac{1}{18} \sum_i \bar{\mathbf{G}}_{T_i}^{\mathrm{T}} \bar{\mathbf{G}}_{T_i} \right) \begin{pmatrix} s \\ t \\ 1 \end{pmatrix} \tag{7}$$

and employ the quadratic minimization procedure described in [24] (and later simplified in [25]) to obtain the optimal texture coordinates. Here f_{T_p} signifies the square of the residual "oriented volume" (if any), and f_{T_o} is the sum of squared changes in volume. As in the case of vertex placement, we compute the new texture coordinates by performing the preservation step first and, if underconstrained, proceed with the optimization step. This gives us a "memoryless" procedure for selecting texture coordinates, and we incorporated this method into our geometry-driven and image-driven simplification algorithms.

4.2 Image-Based Edge Cost

In Section 3.2, we presented an image metric for determining the visual similarity between an original and a simplified model (Equation 2). Our edge cost measure is based on this image metric. That is, edge collapses are ordered such that the edge that yields the smallest *visual difference* is the next edge collapsed. Since we are only interested in how to order the edges, we can use the MS error rather than the RMS error. In this section, we describe a fast method for evaluating the edge cost without having to iterate over the entire triple sum in Equation 2.

4.2.1 Definition of Edge Cost

Let $\mathcal{Y}^0$, $\mathcal{Y}^k$, and $\mathcal{Y}^{k+1}$ be the collections of images of the original model, the current model after k edges have been collapsed, and the model after an additional edge e has been collapsed. We seek to find the edge e that minimizes the mean square error

$$e = \underset{e}{\operatorname{argmin}}\, d_{MS}(\mathcal{Y}^0, \mathcal{Y}^{k+1}) = \underset{e}{\operatorname{argmin}} \frac{1}{lmn} \sum_{h=1}^{l} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(y_{hij}^0 - y_{hij}^{k+1} \right)^2$$

and then collapse it. For reasons that will become apparent, we subtract the constant $d_{MS}(\mathcal{Y}^0, \mathcal{Y}^k)$ and multiply by another constant lmn to obtain an expression for the cost $f(e,k)$ of collapsing edge e in iteration k:

$$\begin{aligned} f(e,k) &= lmn\left(d_{MS}(\mathcal{Y}^0, \mathcal{Y}^{k+1}) - d_{MS}(\mathcal{Y}^0, \mathcal{Y}^k)\right) \\ &= \sum_{h=1}^{l}\sum_{i=1}^{m}\sum_{j=1}^{n}\left[\left(y_{hij}^0 - y_{hij}^{k+1}\right)^2 - \left(y_{hij}^0 - y_{hij}^k\right)^2\right] \end{aligned}$$

It should be clear that the introduction of these additional positive constants does not change the edge order given by the mean square error. Note that any pixel satisfying $y_{hij}^k = y_{hij}^{k+1}$ makes no contribution to $f(e,k)$. In fact, this holds for the majority of pixels due to the spatial locality of the edge collapse. That is, when an edge e is collapsed and replaced with a vertex v, the only pixels that can differ between the images $\mathcal{Y}^k$ and $\mathcal{Y}^{k+1}$ are the ones covered by the triangles $\lceil\lfloor\lceil\lfloor e\rfloor\rceil\rfloor\rceil \cup \lceil\lfloor\lceil v\rceil\rfloor\rceil$. For efficiency, we compute for each view h an axis-aligned bounding box $R_h = I_h \times J_h$ in screen space around these triangles, or equivalently the set of vertices $\lfloor\lceil\lfloor e\rfloor\rceil\rfloor \cup \{v\}$, which is a conservative estimate of the affected pixels. We only visit this smaller set of pixels, which results in an edge cost expression that is faster to evaluate:

$$f(e,k) = \sum_{h=1}^{l}\sum_{i\in I_h}\sum_{j\in J_h}\left[\left(y_{hij}^0 - y_{hij}^{k+1}\right)^2 - \left(y_{hij}^0 - y_{hij}^k\right)^2\right] \tag{8}$$

4.2.2 Evaluation of Edge Costs

In theory, the edge cost $f(e,k)$ would have to be evaluated each iteration k for the entire set of remaining edges. Contrary to most geometry-based edge collapse methods, for which the edge cost does not change with k except for a small set of edges near the previously collapsed edge, an edge collapse in our algorithm could in principle affect the cost of any remaining edge. Even geometrically and topologically distant edges may be arbitrarily near in some views. However, since only a fraction of pixels generally change between iterations, collapsing an edge does not alter the edge costs for most of the other edges. That is, even when the images change between iterations, $f(e,k)$ is independent of k as long as the changes take place outside e's associated bounding box (see Equation 8), which eliminates the need to constantly update each edge cost. Therefore, instead of attempting to identify all affected edges, we have chosen to consider only the set $\lceil\lfloor\lceil v\rceil\rfloor\rceil$, which are the edges updated in our geometry-based method (see also Figure 2). These are generally the only edges whose cost terms change in more than one view. We validated this assumption by recording the rank of the collapsed edges among all valid candidates over a large number of iterations. For the Stanford bunny model, the cost of the collapsed edge was on average in the lowest 0.1%.

As in most edge collapse algorithms, collapsing an edge generally leads to an increase in the cost of collapsing any of the nearby affected edges. This suggests that the algorithm is amenable to *lazy evaluation*, that is, instead of updating the cost of an affected edge, the edge is marked as "dirty" and the evaluation is deferred until the edge reaches the front of the queue [6]. When a dirty edge is removed from the queue, its edge cost is re-evaluated, and the edge is re-inserted into the queue. Without lazy evaluation, each edge generally has its cost updated several times before it reaches the front of the queue. By employing lazy evaluation, many of these extraneous evaluations are eliminated. Lazy evaluation generally reduces the number of edge cost evaluations by a factor of five (depending on the degree of simplification), without significantly compromising model quality. For the same test case as above, lazy evaluation degraded the average percentile of the collapsed edges from 0.1% to 5%. This lazy evaluation scheme was used in our image-driven algorithm for all results presented in this paper.

4.2.3 Fast Image Updates

So far, we have described how to evaluate $f(e,k)$ given sets of images $\mathcal{Y}^0$, $\mathcal{Y}^k$, and $\mathcal{Y}^{k+1}$. However, we have not explained how these images, or subimages, are generated. Our main approach has been to maintain images $\mathcal{Y}^k$ of the current model in the simplification, and to make incremental changes to these images to produce the images used in the next iteration, as well as the images $\mathcal{Y}^{k+1}$ used to evaluate the cost of collapsing e. Our implementation is based on *OpenGL* and uses the *pixel buffer* extension for hardware-assisted off-screen rendering.

Our algorithm begins by rendering images $\mathcal{Y}^0$ of the original model and stores these away. Because many models are greatly over-tessellated, one may use a faster, geometry-driven algorithm to produce a coarser model with little or no loss in quality, and then begin the image-driven simplification after $k \geq 0$ edges have been collapsed. In addition to the original images, we render images $\mathcal{Y}^k$ of the current model which, depending on the initial value of k, may or may not be the same as the original images. The images $\mathcal{Y}^k$ are rendered from scratch only in the first iteration of the algorithm, and are then updated between iterations via small local changes.

One of the most fundamental premises of the algorithm is the ability to quickly produce a set of images $\mathcal{Y}^{k+1}$ from existing images $\mathcal{Y}^k$ without having to re-render the entire mesh. More specifically, we require an algorithm for replacing the rendered triangles $T^k = \lceil\lceil\lfloor e\rfloor\rceil\rceil$ with a set of new triangles $T^{k+1} = \lceil\lceil v\rceil\rceil$ in all l views. On average, $|T^k| = 10$ and $|T^{k+1}| = 8$, so the number of triangles involved is fairly small relative to the thousands or even millions of triangles in the original model. Thus, we would like an algorithm for "un-rendering" the triangles T^k, revealing any obscured parts of the surface, and then rendering the replacement triangles T^{k+1}. In a software renderer, this can be accomplished by using an *object buffer*—a multi-layer frame buffer that maintains not only the top, visible surface layer, but a depth-sorted list of all surface fragments that project into a pixel [1]. While attractive in its simplicity, the object buffer is not commonly available in hardware, and the low fill-rate afforded by a software-based renderer greatly limits the simplification speed. In the remainder of this section, we will describe how hardware rendering can be used to accelerate the simplification.

Even though un-rendering is not supported in hardware, we can limit the number of triangles that have to be rendered to produce $\mathcal{Y}^{k+1}$. We exploit the spatial locality of the edge collapse by maintaining a triangle "bucket" for each pixel row and column (Figure 4). The buckets are implemented as hash tables that are indexed by the triangle identifiers. This subdivision allows us to efficiently cull away most triangles that do not intersect the rectangular region $R_h = I_h \times J_h$ (see Section 4.2.1) in each view h, leaving the set T_{R_h}. Since the procedure is the same for all views h, we will omit the subscript h in the following paragraph for the sake of readability.

Triangle culling is accomplished by computing the union of the vertical buckets $T_I = \cup_{i\in I} T_i$ and the horizontal buckets $T_J = \cup_{j\in J} T_j$ spanned by R, and then letting $T_R = T_I \cap T_J$ be a conservative (but generally tight) estimate of the set of triangles contained in R. As these hash tables can get quite large, we accelerate the computation of unions by maintaining an additional set of tables $\Delta T_i = T_i \setminus T_{i-1}$ that are the set differences between consecutive pixel columns (and similarly for the horizontal buckets). That is, ΔT_i contains the triangles whose left-most vertex is in column i. Then we can rewrite T_I as $T_{\min I} \cup \Delta T_{\min I+1} \cup \cdots \cup \Delta T_{\max I}$, with the property that the sets in this union are pair-wise disjoint. In general, the hash tables ΔT_i are considerably smaller than the tables T_i. The intersection T_R can then be computed in linear time by associating a "time stamp" with each triangle. Prior to computing T_R, a unique time stamp is chosen. While building the set T_I, all triangles encountered are marked with the new time stamp. As T_J is traversed, only the triangles with the given time stamp are added to T_R. Even though the sets T_I and T_J may be much larger than their intersection, this simple image partitioning scheme has

proven very efficient in practice. Compared to an implementation that uses a k-d tree partitioning of each image, the bucket-based implementation described in this section runs two to three times faster.

When we wish to replace T^k with T^{k+1}, we first clear each region R_h in which these sets of triangles are contained. We then render the triangles $T_{R_h} \setminus T^k$, i.e. all triangles (both visible and occluded) in R_h except those we wish to un-render. We complete the edge collapse by rendering the set T^{k+1}, producing the images $\mathcal{Y}^{k+1}$, which then allows us to calculate the edge cost $f(e, k)$.

5 RESULTS

In this section we present several models that have been simplified using our image-driven technique. All models presented in this paper were simplified on a 250 MHz R10000 Silicon Graphics Octane workstation with 256 MB of RAM and IMPACTSR graphics. As mentioned in Section 4.2.3, we pre-simplified all of the models using our geometry-driven "memoryless" method [24, 25], and used the image-based cost measure for the remainder of the simplification. Table 1 contains the number of triangles eliminated and the time spent in this pre-simplification step. This table also provides data for the image-driven step, as well as total simplification time and effective triangle reduction rates. We point out that while our image-driven algorithm is slow in comparison with the fastest geometry-based simplification methods currently available, it still produces acceptable reduction rates (roughly in the range of 100 to 300 triangles per second) when pre-simplification is used. Since our edge collapse algorithm can produce a progressive mesh [18], the expense of simplification needs only be paid once—a price that is often trivial compared to the entire modeling pipeline, which in addition to simplification may require days of manual and computational labor for large models.

5.1 Image versus Geometric Method

Because the idea of using images rather than geometry to guide simplification is such a large departure from mainstream simplification methods, we will first compare objects that are created by this new method against those of two other simplification techniques. We created several levels-of-detail of the Stanford Bunny using our image-driven method, our purely geometry-driven method [24, 25], and the quadric technique of Garland and Heckbert (QSlim) [13, 14].[1] We chose these two other methods for the high quality of their results and because the code for both are available to us.

Color Plate 1 shows two views of the original bunny model, a memoryless simplification model, and an image-driven model. The bunny models produced by these two methods are comparable in terms of visual quality. There are, however, some subtle differences between the results of these two methods that we have observed for a variety of flat shaded models. First, the image-driven method generally pays more attention to preserving the shape of silhouettes and sharp edges (see, for example, the shape of the bunny ears and the rounding along and across the edges of the feet). Second, soft interior edges with small intensity gradients, such as small bumps and furrows, are typically smoothed out and replaced with larger triangles by the image-driven method (e.g. the chest and the back of the bunny). As a rule of thumb, the image-driven method spends relatively more detail preserving the sharp intensity boundaries in an image, which convey most of the large-scale shape of an object. This is typically the relevant information one seeks to retain in a highly simplified model. Third, the image-driven method better balances the ratio of boundary edges to interior edges of a model by using a single unified error metric, as evidenced by

[1] For the bunny model, we used QSlim v2.0 with the options `-B 10`. For the salamander and torus models, we used PropSlim v1.2.

model	*options*	*triangles*			*simplification time*			*tri/s*
		original	*after pre-simp.*	*final*	*pre-simp.*	*image-driven*	*total*	
bunny		69,451	30,254	2,899	0:47	28:56	29:43	37
			14,535	1,333	0:57	11:18	12:15	93
			6,444	654	1:01	4:38	5:39	203
			2,918	304	1:04	2:20	3:24	339
			1,336	143	1:05	1:28	2:33	453
turbine blade	20/20	1,765,388	32,767	8,191	26:45	3:28:10	3:54:55	125
	10/20		32,767	8,191	26:45	1:06:49	1:33:34	313
	20/20		8,191	2,046	27:06	23:25	50:31	582
dragon	flat	871,306	32,768	8,191	12:15	38:24	50:39	284
	smooth		32,768	8,191	12:15	1:14:12	1:26:27	166
buddha	flat	1,087,716	65,536	16,384	15:19	1:57:49	2:13:08	134
	smooth		65,536	16,384	15:19	2:13:27	2:28:46	120
salamander	orange	71,312	16,384	2,047	1:21	15:52	17:13	67
	green		16,384	2,047	1:21	16:09	17:30	66
torus		65,536	4,096	512	1:05	4:28	5:33	195

Table 1: Simplification results for the image-driven method. For the turbine blade model, the *options* column indicates the ratio of the number of exterior views to the total number of views used during simplification. For the dragon and buddha models, the options indicate the choice of shading interpolation, while the choice of texture used during simplification is given for the salamander model. All timings correspond to number of wall clock hours, minutes, and seconds. The last column gives the effective number of triangles eliminated per second, i.e. the difference in number of triangles of the original and final models over the total simplification time.

Plates 1d–f. Most geometry-based algorithms have difficulties choosing an appropriate level of boundary fidelity without explicit user intervention, e.g. via a parameter that allows boundary and interior edges to be weighted differently. The image-driven method, on the other hand, accounts for the visual impact due to boundary changes directly. Finally, the image-driven method often produces better shaped triangulations that more closely follow the intrinsic geometry of a surface, with few sliver triangles and other artifacts such as folds in the mesh. Such degeneracies are visually disturbing and are easily caught by the image metric.

We made two quantitative measurements of the levels-of-detail that were produced. First, we used the geometric tool Metro to calculate the mean geometric error between each of the simplified models and the original [4].[2] Figure 5 is a graph of the results, where the errors are expressed in units of $1/50^{th}$ of the bounding box diagonal of the original model. The memoryless method produces models that have the least geometric error, followed closely by the image-driven method, and then the quadric method. Using the image cost produces models that are surprisingly close to the original models geometrically. We attribute this result to three factors: 1) the use of the high quality memoryless algorithm to pre-simplify the models and as the method of positioning vertices in the image-driven method, 2) the indirect effects of preserving the shading of a model, which depends directly on the model's geometric shape, and, more importantly, 3) the sensitivity of the RMS image metric to silhouette discrepancies. For progressively coarser models, the likelyhood of an edge being on a silhouette increases (see Figure 7). Thus, given a large number of views,

[2] We used Metro v2.5 with the options `-s -t`.

we expect the majority of edges to be silhouette edges from several viewpoints, which limits the amount of geometric degradation.

The second form of evaluation that we performed used an image metric. Each model was rendered from 24 camera positions, as described in Section 3.2, and the RMS error between the original and simplified model's images were calculated. These errors are the same as the d_{RMS} values provided in the captions. Figure 6 shows the results of this. As expected, the image-driven method consistently produced models of higher quality than the two other methods with respect to this measure. We note that there are three ways in which this image metric differs in detail from the actual metric used during the simplification process. First, during simplification the method relies on rendering hardware to perform polygon scan conversion, and we did not use anti-aliased images. In contrast, the images that were used to calculate the evaluation of Figure 6 were anti-aliased images that were created using the RenderMan program. Second, the dimensions of the images used during simplification were 256×256 pixels, while the final images have dimensions 512×512. Finally, the distribution and number of camera positions on the sphere were different during simplification than evaluation (20 versus 24).

Is it a tautology that a model produced by image-driven simplification is a close match to the original model when evaluated using an image metric? No more so, we argue, than using geometric proximity to guide simplification and then measuring geometric differences. More important is the decision of what result is desired—geometric similarity or visual similarity. We believe that visual similarity is often the goal of simplification, and that we should try to achieve this goal in a direct manner, using images.

5.2 Simplification of Hidden Interiors

Image-driven simplification removes detail that has little effect on rendered images of a model. Consequently, if a portion of a model is completely invisible in all views, it will be radically simplified. Color Plates 2 and 3 provide an example of this. The original model (Plate 2b) is a turbine blade that has a complex arrangement of internal cavities that accounts for much of the model's 1,765,388 polygon complexity. Plate 3b shows the same model with the outer surface transparent so that this interior detail is visible. When simplified using a geometry-driven method all of this interior detail is retained, yet none of it contributes to opaque images of the model (Plates 2a, 2d, and 3a). The image-driven method, on the other hand, produces a model that has a more detailed exterior and a greatly simplified interior (Plates 2c, 2f, and 3c). Removing all of the internal detail allowed a substantially higher polygon budget to represent the visible portions of the model, such as the numerous holes on the left side of the blade, as illustrated by the closeups in Plate 2e. Due to the topological complexity of this model, we allowed both methods to simplify the surface topology by placing no restrictions on what edges could be collapsed. This is the reason why many of the smaller holes were closed by the geometry-based method.

The improvement in exterior appearance provided by the image-driven method clearly comes at the expense of drastically eliminating interior detail. This tradeoff is desirable if the model is only to be viewed from the outside. We envision, however, that some users may want more control over the tradeoff between interior and exterior quality such that hidden parts, while being greatly simplified, at least retain their basic shape. For models with low depth complexity, we have devised a partial solution by allowing the exterior surface to be removed in a subset of the views during simplification. We achieve this by enabling front-face culling in OpenGL, which comes at no additional computational expense. Plate 3d shows the same blade model for which the interior was revealed in half of the 20 views. Consequently, the interior of this model is much better retained, while the exterior quality suffered slightly. In this manner, the user is given smooth control over the balance between interior and exterior detail by choosing the number of interior and exterior views appropriately.

We acknowledge that this solution is not universal, and that there exist models for which additional steps may have to be taken for applications that demand preservation of fully or partially hidden details. As part of our future research, we plan to investigate how the use of translucency, cutting planes, model segmentation into independent parts, and optimal view coverage (such as camera positions inside the model) can be used to further improve our algorithm. We see this issue as secondary, however, and return our focus to the goal of producing high quality simplifications of the visible parts of a model.

5.3 Flat and Gouraud Shading

Geometric simplification methods do not take into account the type of shading interpolation that will be used when a model is rendered. None of the published methods make a distinction between objects that will be flat shaded (constant shading over each face) and those that will be rendered with Gouraud interpolation (linear interpolation of the vertex intensities). Image-driven simplification, on the other hand, can account for these differences during simplification.

We chose to compare flat and Gouraud shading since these are the two shading models most commonly supported in both software and hardware. While Phong interpolation usually yields higher quality shading, it is seldom available in graphics hardware, and is rarely applied to coarse polygonal objects as its high computational expense would offset any performance gains attained by the use of simplified models.

Color Plate 4 shows a dragon model that was simplified twice with our image-driven approach. The first simplified model was guided by images in which the intermediate models were rendered using diffuse flat shading of the faces (Plates 4a and 4d). The second simplification was guided by images for which specular reflection and Gouraud interpolation of the intensities calculated at the polygon vertices were used (Plates 4c and 4f). For the Gouraud shaded models, the vertex normals were computed as the area-weighted sum of incident face normals.[3]

Plates 4a–f show each of these two simplified models and the original model rendered with flat and with Gouraud interpolation. The model produced with flat shading is the better of the two when viewed with flat shading, and the Gouraud-image-driven model retains much more of the detail in the Gouraud-rendered image. The lower-right dragon retains better details in many places, including the front scales, the face, the feet, the hind leg and the ridge along the back (see Plates 5–7 for closeups of these features). The reason this detail is kept is because the simplification process takes into account the effect of all the various surface attributes during the simplification process. The quality tradeoff between polygon size, vertex positions, and variation in surface normals across triangles are all handled by the image metric. Plates 8 and 9 further demonstrate how different shading models can successfully be accounted for.

If the rendering parameters are not known beforehand, one might suspect that the models produced by the image-driven method would be greatly suboptimal when displayed using a different set of rendering parameters. In practice, however, we have found that the image-driven method usually outperforms the geometry-driven methods even when the rendering parameters do not match. To optimize the model for a variety of rendering conditions, one could alternatively vary the parameters between views during simplification (cf. Section 5.2), and thus produce an "average" model that is not tailored to a particular set of rendering or surface parameters.

[3]Note that since the normals for the vertices $\lfloor\lceil v\rceil\rfloor$ depend on the position of v, so does the shading of the triangles $\lceil\lceil\lfloor\lceil v\rceil\rfloor\rceil\rceil$. Thus, for Gouraud shading, the sets T^k and T^{k+1} must be expanded accordingly (see Section 4.2.3).

5.4 Texture-Sensitive Simplification

Published methods for preserving texture coordinates on simplified models have concentrated on avoiding shifts in texture coordinates across the *entire surface* of a simplified model. Many textures, however, vary in the amount of color variation over different portions of a model. Texture of a human face is nearly uniform over much of the skin, but varies greatly near the eyes, and a map of the Earth is detailed on the continents but is uniform over the oceans. These differences in color detail provide an opportunity for heavier simplification in the regions of nearly uniform color. We present the results of simplifying two models with natural looking textures, and one artificial model with a more challenging texture content.

The image-driven simplification method easily recognizes and exploits the opportunity of varying texture detail. Plates 10c and 11c show two textured versions of the same geometric salamander model, one with thin black spots on orange and another with large black spots on green. The underlying polygon model and the texture coordinates are the same for both original models. This model was simplified once with QSlim and the memoryless geometric method, which both ignore the actual texture image used, and twice with the image-driven method, once with each texture. When the image-driven models are rendered with the same texture as when they were created, they closely match the originals. When the textures are swapped, however, the spots shift and change shape. This can be seen in the closeups and difference images with respect to the original model that are given for each simplified salamander. This indicates that the simplified models devoted only those polygons necessary to maintain the texture detail, and did not expend additional polygons in the uniformly colored regions of the texture. The geometrically simplified salamander has texture distortions with either texture. In particular, both of the geometry-driven algorithms had great difficulties preserving the texture along the underside of the salamander, where the texture wraps around and meets itself along a jagged seam (see the closeups in Plate 11).[4] The image-driven method, on the other hand, had no difficulties accounting for distortion along the seam, and consequently produced more visually pleasing results.

The last model used in our comparison was constructed with a highly non-linear texture parameterization (thereby making it more difficult to preserve using a piece-wise linear approximation) by sweeping and spinning an ellipse around a circle to form a torus (Plate 12). The texture image consists of randomly colored square blocks on a white background, with a number of straight black lines and several rows of text. We anticipated that distortion to such familiar geometric shapes would be easily detectable by the human eye. Plates 12a–c show the simplified models. Again, the image-driven method outperformed the geometry-based ones, both qualitatively and numerically, as given by the image metric. As highlighted by the difference images, the image-driven method generally distorted the texture less than a couple of pixels in any direction, resulting in very little perceivable loss in texture quality, which can be seen to be much more noticeable in the other two models.

Just as geometric fidelity of a simplified model depends on the methods used for placing new vertices and ordering edge collapses, the visual quality of a texture parameterization over a simplified surface is determined by two factors: the method for computing per-vertex texture coordinates, and the order in which edges are collapsed. We have found that our "memoryless" approach to computing texture coordinates is a good alternative for the first of these two components. Even when the edge collapses are ordered without regard to texture quality, as in our geometry-driven method, the results are generally better than those produced by QSlim. When texture appearance is additionally accounted for in the edge cost, as in our image-driven algorithm, the increase in visual quality can be quite dramatic.

[4] In contrast to our geometry- and image-driven algorithms, QSlim does not support multiple texture coordinate pairs per vertex, and so had additional problems dealing with the texture seams.

6 CONCLUSIONS AND FUTURE WORK

We presented a simplification method that is novel both in its use of images to evaluate edge costs and also in using graphics acceleration hardware to make these evaluations. This approach unifies decisions about different aspects of a model such as vertex positions, colors, normals and texture. Specific characteristics of the method include:

- Produces models of high visual and geometric fidelity.
- Accounts for the shading interpolation method being used.
- Eliminates details in hidden portions of a model.
- Creates models that are sensitive to surface texture.

We introduced a new method for assigning texture coordinates to the vertices of a simplified model. This technique is independent of the method used for placing the new vertex after an edge collapse, and has proven useful both in our geometry- and image-driven simplification algorithms. Even without an image metric to guide the simplification, our texture coordinate mapping generally results in less texture distortion than the method recently proposed by Garland and Heckbert [14].

One exciting direction for future work is the exploration of image metrics that are better tuned to the human visual system. There is a wealth of information in the psychophysical literature that can be drawn upon, and several promising perceptually-motivated image metrics already exist, including the "Visible Differences Predictor" [8] and the Sarnoff Model [26]. We might also draw upon the work in computer graphics that use image metrics, which include [12, 28, 30, 33]. We are particularly interested in the work of Bolin and Meyer, in which they define a perceptually-based metric that is also fast to calculate [2]. We view our work as only a first step towards bridging the gap between model simplification and knowledge about human visual perception.

Another intriguing issue is whether the placement of new vertices may be guided by an image metric. The challenge is to find a fast way to place a vertex that does not invoke a slow gradient descent process to find the optimal position.

References

[1] ATHERTON, P. R. A Method of Interactive Visualization of CAD Surface Models on a Color Video Display. Proceedings of SIGGRAPH 81. In *Computer Graphics* 15(3) (August 1981), pp. 279–287.

[2] BOLIN, M., MEYER, G. A Perceptually Based Adaptive Sampling Algorithm. Proceedings of SIGGRAPH 98. In *Computer Graphics* Proceedings, Annual Conference Series, 1998, ACM SIGGRAPH, pp. 299–309.

[3] CIAMPALINI, A., CIGNONI, P., MONTANI, C., and SCOPIGNO, R. Multiresolution Decimation Based on Global Error. *The Visual Computer*, Springer International, 13(5), 1997, pp. 228–246.

[4] CIGNONI, P., ROCCHINI, C., and SCOPIGNO, R. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum*, 1998 (in press)

[5] COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., BROOKS, F., and WRIGHT, W. Simplification Envelopes. Proceedings of SIGGRAPH 96. In *Computer Graphics* Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 119–128.

[6] COHEN, J., MANOCHA, D., and OLANO, M. Simplifying Polygonal Models Using Successive Mappings. In *IEEE Visualization '97* Proceedings, October 1997, pp. 395–402.

[7] COHEN, J., MANOCHA, D., and OLANO, M. Appearance-Preserving Simplification. Proceedings of SIGGRAPH 98. In *Computer Graphics* Proceedings, Annual Conference Series, 1998, ACM SIGGRAPH, pp. 115–122.

[8] DALY, S. The Visible Difference Predictor: An Algorithm for the Assessment of Image Fidelity. In *Digital Images and Human Vision*, edited by A. B. Watson, MIT Press, 1993, pp. 179–206.

[9] EL-SANA, J. and VARSHNEY, A. Generalized View Dependent Simplification. Proceedings of Eurographics 99. In *Computer Graphics Forum*, 18(3), September 1999, pp. 83–94.

[10] EL-SANA, J., AZANLI, E., and VARSHNEY, A. Skip Strips: Maintaining Triangle Strips for View-Dependent Rendering. In *IEEE Visualization '99* Proceedings, October 1999, pp. 131–138.

[11] ERIKSON, C. and MANOCHA, D. GAPS: General and Automatic Polygonal Simplification. In *ACM Symposium on Interactive 3D Graphics '99* Proceedings, April 1999, pp. 79–88.

[12] FERWERDA, J. A., PATTANAIK, S. N., SHIRLEY, P., and GREENBERG, D. P. A Model of Visual Masking for Computer Graphics. Proceedings of SIGGRAPH 97. In *Computer Graphics* Proceedings, Annual Conference Series, 1997, ACM SIGGRAPH, pp. 143–152.

[13] GARLAND, M. and HECKBERT, P. S. Surface Simplification using Quadric Error Metrics. Proceedings of SIGGRAPH 97. In *Computer Graphics* Proceedings, Annual Conference Series, 1997, ACM SIGGRAPH, pp. 209–216.

[14] GARLAND, M. and HECKBERT, P. S. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In *IEEE Visualization '98* Proceedings, October 1998, pp. 263–269.

[15] GORTLER, S., GRZESZCZUK, R., SZELISKI, R., and COHEN, M. The Lumigraph. Proceedings of SIGGRAPH 96. In *Computer Graphics* Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 43–54.

[16] GU, X., GORTLER, S., HOPPE, H., MCMILLAN, L., BROWN, B., and STONE, A. Silhouette Mapping. Technical Report TR–1–99, Department of Computer Science, Harvard University, March 1999.

[17] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., and STUETZLE, W. Mesh Optimization. Proceedings of SIGGRAPH 93. In *Computer Graphics* Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 19–26.

[18] HOPPE, H. Progressive Meshes. Proceedings of SIGGRAPH 96. In *Computer Graphics* Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 99–108.

[19] HOPPE, H. View-Dependent Refinement of Progressive Meshes. Proceedings of SIGGRAPH 97. In *Computer Graphics* Proceedings, Annual Conference Series, 1997, ACM SIGGRAPH, pp. 189–198.

[20] HOPPE, H. New Quadric Metric for Simplifying Meshes with Appearance Attributes. In *IEEE Visualization '99* Proceedings, October 1999, pp. 59–66.

[21] KOBBELT, L., CAMPAGNA, S., and SEIDEL, H-P. A General Framework for Mesh Decimation. In *Graphics Interface '98* Proceedings, June 1998, pp. 43–50.

[22] LEVOY, M. and HANRAHAN, P. Light Field Rendering. Proceedings of SIGGRAPH 96. In *Computer Graphics* Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 31–42.

[23] LÉVY, B. and MALLET, J.-L. Non-Distorted Texture Mapping for Sheared Triangulated Meshes. Proceedings of SIGGRAPH 98. In *Computer Graphics* Proceedings, Annual Conference Series, 1998, ACM SIGGRAPH, pp. 343–352.

[24] LINDSTROM, P. and TURK, G. Fast and Memory Efficient Polygonal Simplification. In *IEEE Visualization '98* Proceedings, October 1998, pp. 279–286.

[25] LINDSTROM, P. and TURK, G. Evaluation of Memoryless Simplification. In *IEEE Transactions on Visualization and Computer Graphics*, 5(2), April–June 1999.

[26] LUBIN, J. A Visual Discrimination Model for Imaging System Design and Evaluation. *Vision Models for Target Tracking and Recognition*, edited by Eli Peli, World Scientific, New Jersey, 1195, pp. 245–283.

[27] LUEBKE, D. and ERIKSON, C. View-Dependent Simplification of Arbitrary Polygonal Environments. Proceedings of SIGGRAPH 97. In *Computer Graphics* Proceedings, Annual Conference Series, 1997, ACM SIGGRAPH, pp. 199–208.

[28] MACIEL, P. W. C. and SHIRLEY, P. Visual Navigation of Large Environments Using Textured Clusters. *1995 Symposium on Interactive 3D Graphics*, Monterey, California, April 9–12, 1995, pp. 95–102.

[29] MCMILLAN, L. and BISHOP, G. Plenoptic Modeling: An Image-Based Rendering System. Proceedings of SIGGRAPH 95. In *Computer Graphics* Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH, pp. 39–46.

[30] RAMASUBRAMANIAN, M., PATTANAIK, S. N., and GREENBERG, D. P. A Perceptually Based Physical Error Metric for Realistic Image Synthesis. Proceedings of SIGGRAPH 99. In *Computer Graphics* Proceedings, Annual Conference Series, 1999, ACM SIGGRAPH, pp. 73–82.

[31] RONFARD, R. and ROSSIGNAC, J. Full-Range Approximation of Triangulated Polyhedra. Proceedings of Eurographics 96. In *Computer Graphics Forum*, 15(3), August 1996, pp. 67–76.

[32] ROSSIGNAC, J. and BORREL, P. Multi-Resolution 3D Approximations for Rendering Complex Scenes. In *Modeling in Computer Graphics*, edited by B. Falcidieno and T. L. Kuuni, Springer-Verlag, 1993, pp. 455–465.

[33] RUSHMEIER, H., WARD, G., PIATKO, C., SANDERS, P., and RUST, B. Comparing Real and Synthetic Images: Some Ideas About Metrics. *Proceedings of the* 6^{th} *Eurographics Workshop on Rendering*, June 1995, pp. 213–222.

[34] SCHROEDER, W. J., ZARGE, J. A., and LORENSEN, W. E. Decimation of Triangle Meshes. Proceedings of SIGGRAPH 92. In *Computer Graphics* 26(2) (July 1992), pp. 65–70.

[35] SCHROEDER, W. J. A Topology Modifying Progressive Decimation Algorithm. In *IEEE Visualization '97* Proceedings, October 1997, pp. 205–212.

[36] SILVA, C., AT&T Labs-Research. Personal correspondence, November 1999.

[37] XIA, J. and VARSHNEY, A. Dynamic View-Dependent Simplification for Polygonal Models. In IEEE VISUALIZATION '96 Proceedings, October 1996, pp. 327–334.

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

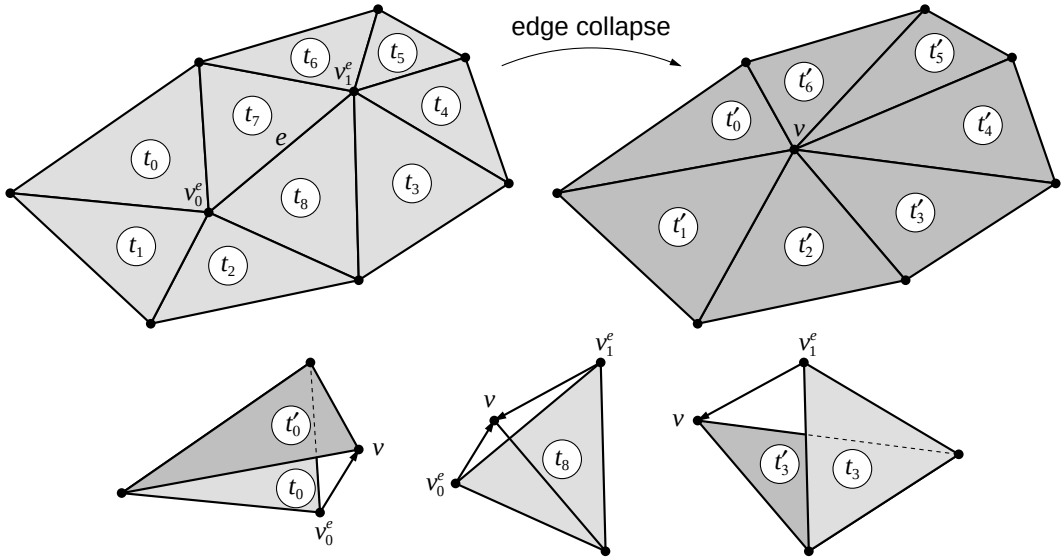


Figure 1: The edge collapse operation. The edge e is collapsed and replaced with a vertex v, removing triangles t_7 and t_8. Example tetrahedral volumes associated with triangles t_0, t_3, and t_8 are shown.

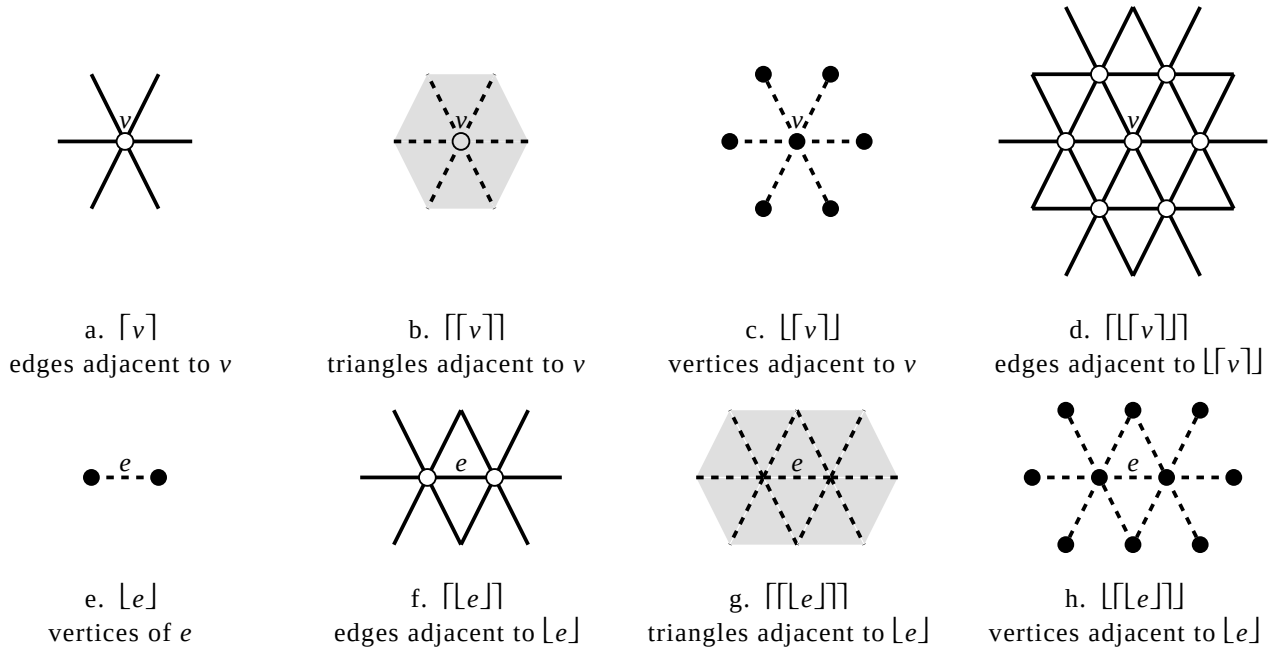


Figure 2: The simplex operators $\lfloor s \rfloor$ and $\lceil s \rceil$.

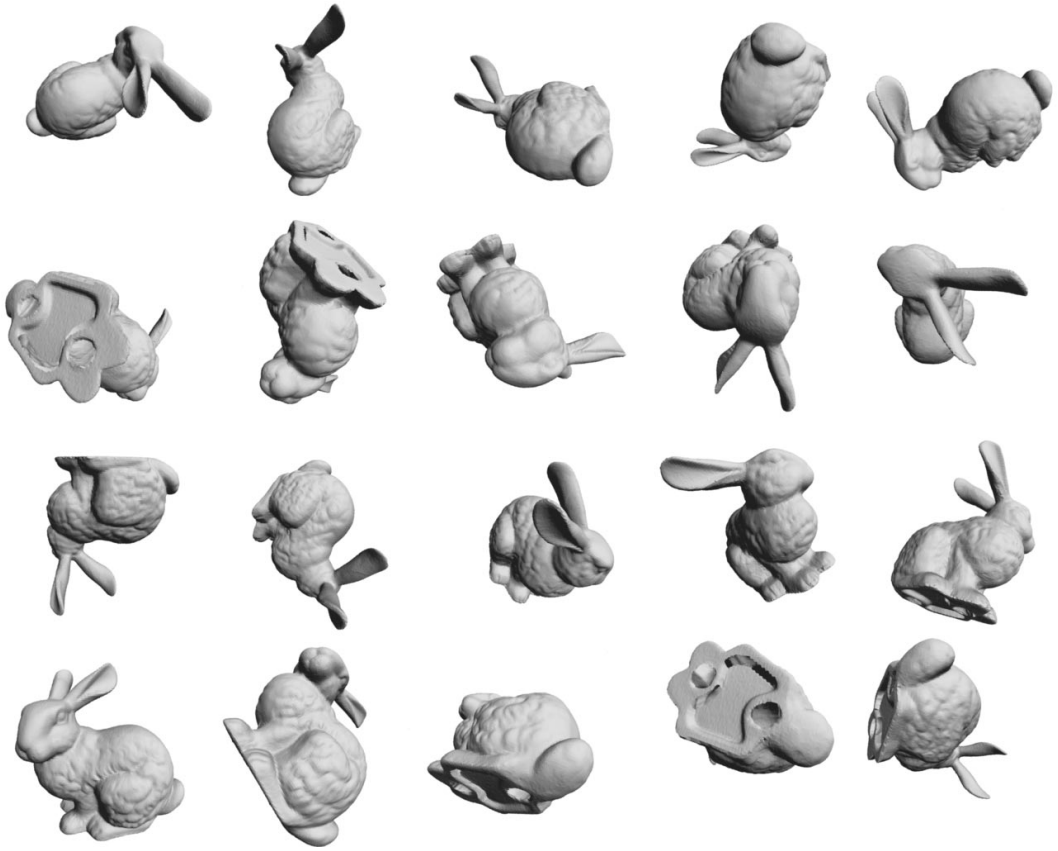


Figure 3: The twenty different views of an object used during simplification. The viewpoints are distributed uniformly and correspond to the vertices of a regular dodecahedron.

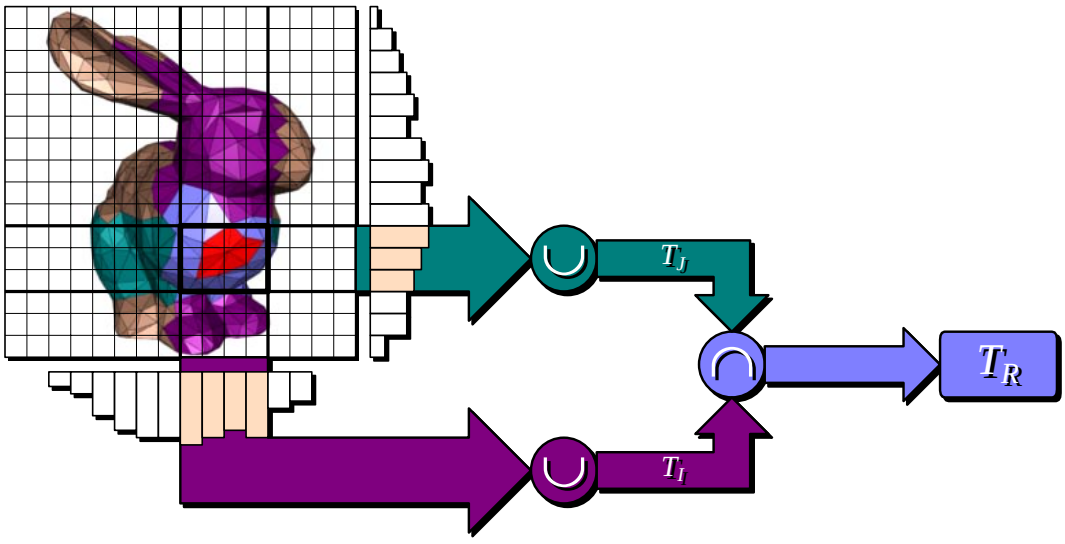


Figure 4: The triangle bucket data structure. The triangles of the model are projected onto the two image axes and are maintained in hash tables for all pixel rows and columns. This data structure allows for all triangles T_R (shown in blue) that intersect the rectangular region R surrounding the triangles T^{k+1} (red) to be accessed quickly. We find T_R by computing the intersection of the triangles T_I (magenta) and the triangles T_J (green).

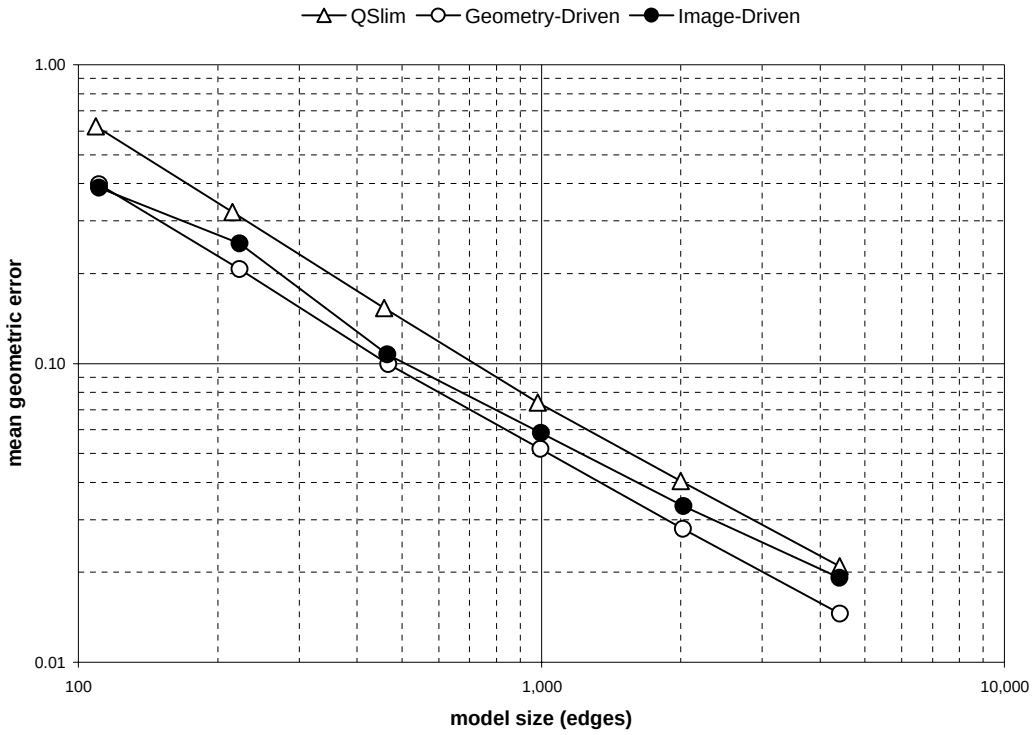


Figure 5: Mean geometric errors for the bunny model.

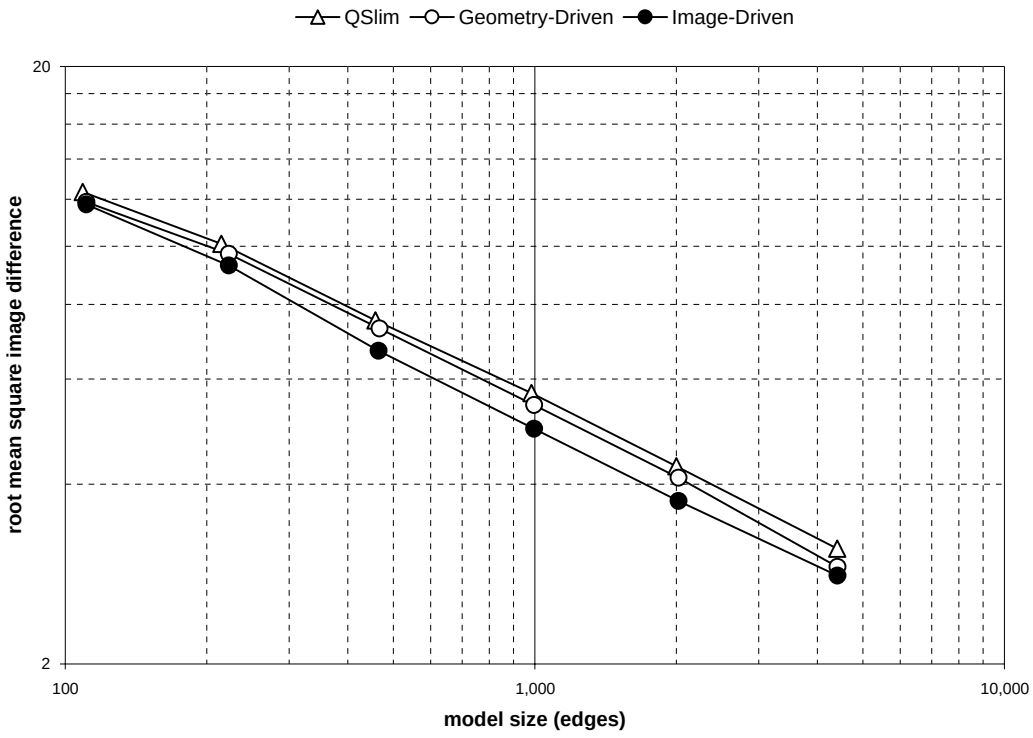


Figure 6: Root mean square image differences for the bunny model. Due to the severe degree of simplification, the coarsest two levels-of-detail for all three methods are of very poor visual quality, which explains the convergence of image differences among the methods.

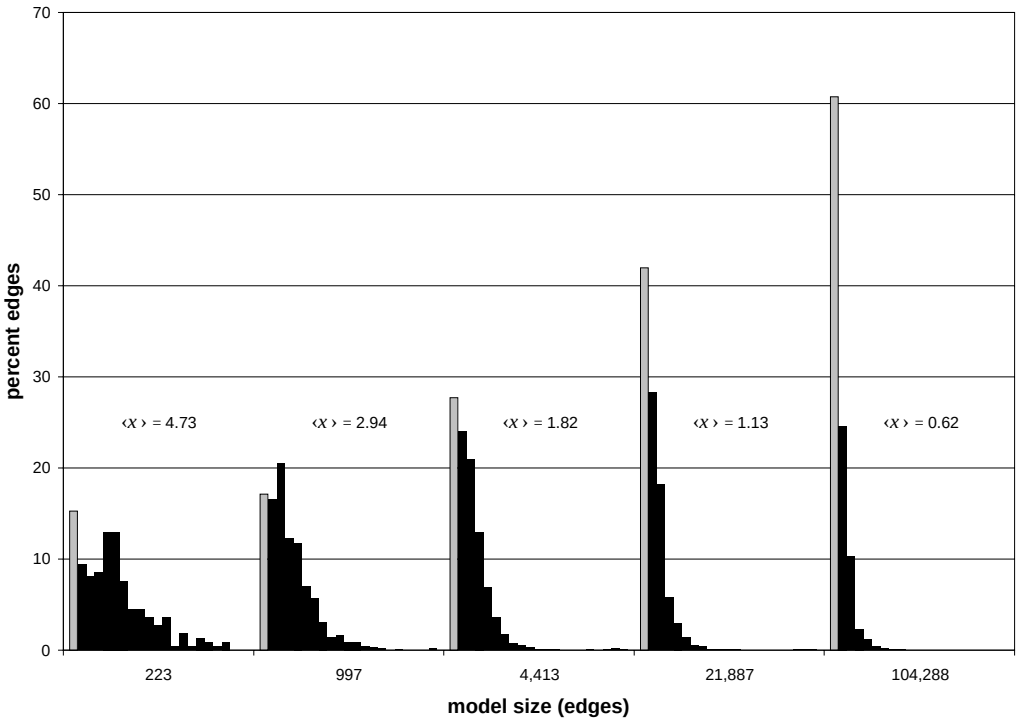


Figure 7: Silhouette edge histograms for the bunny model. For each of the five levels-of-detail, each histogram column corresponds to a certain number of views between zero (leftmost, shaded column) and twenty. The height of the i^{th} column gives the percentage of edges that are silhouette edges in exactly i views. Thus the shaded columns correspond to the percentage of edges that are not on the silhouette in any view. For each level-of-detail, the expectation value $\langle x \rangle$ gives the expected number of views in which an edge is a silhouette edge.

1a. Geometry−driven simplification.
T = 1,336. time = 1:05. d_{RMS} = 4.10.

1b. Original bunny model.
T = 69,451.

1c. Image−driven simplification.
T = 1,333. time = 12:15. d_{RMS} = 3.75.

1d. Geometry−driven simplification.
42 boundary edges.

1e. Original bunny model.
223 boundary edges.

1f. Image−driven simplification.
59 boundary edges.

2a. Geometry−driven simplification. T = 2,048. time = 26:59. d_{RMS} = 3.31.

2b. Original turbine blade model. T = 1,765,388.

2c. Image−driven simplification. T = 2,046. time = 50:31. d_{RMS} = 2.50.

2d. Geometry−driven simplification. T = 8,191. time = 27:06. d_{RMS} = 2.16.

2e. Closeup of geometry−driven simplification (left), original model (middle), and image−driven simplification (right).

2f. Image−driven simplification. T = 8,191. time = 3:54:55. d_{RMS} = 1.57.

3a. Geometry−driven simplification.
T = 8,191. time = 27:06. d_{RMS} = 2.16.

3b. Interior view of original turbine blade model.
T = 1,765,388.

3c. Image−driven simplification.
T = 8,191. time = 3:54:55. d_{RMS} = 1.57.

3d. Image−driven simplification using front−face culling in 10 of 20 views.
T = 8,191. time = 1:33:34. d_{RMS} = 2.10.

4a. Flat shaded model optimized for flat shading.
T = 8,191. time = 50:39. d_{RMS} = 2.67.

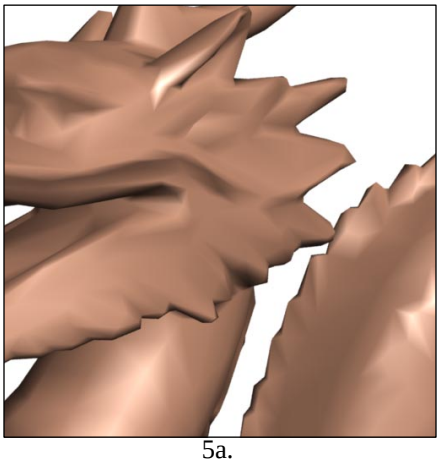
4b. Flat shaded image of original dragon model.
T = 871,306.

4c. Flat shaded model optimized for Gouraud shading.
T = 8,191. time = 1:26:27. d_{RMS} = 3.36.

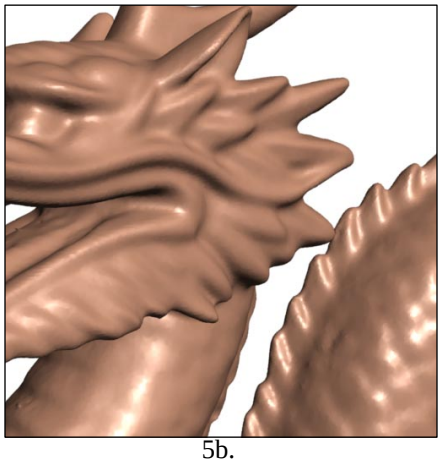
4d. Gouraud shaded model optimized for flat shading.
d_{RMS} = 3.46.

4e. Gouraud shaded image of original dragon model.

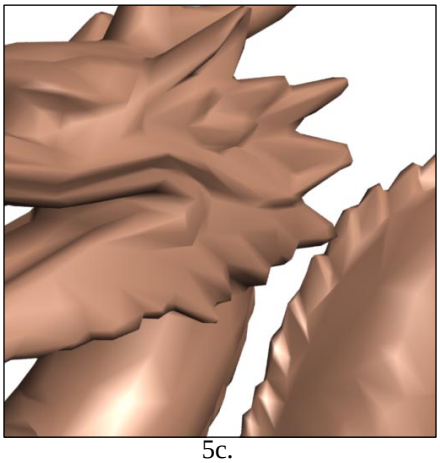
4f. Gouraud shaded model optimized for Gouraud shading.
d_{RMS} = 2.86.



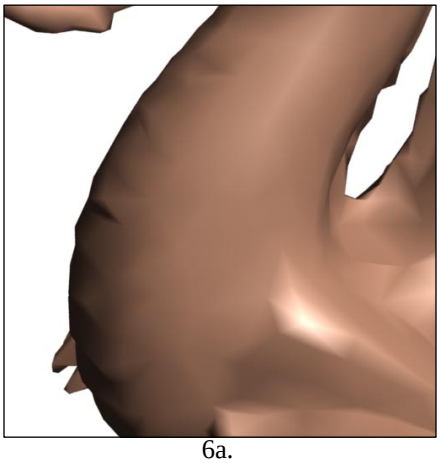
5a.



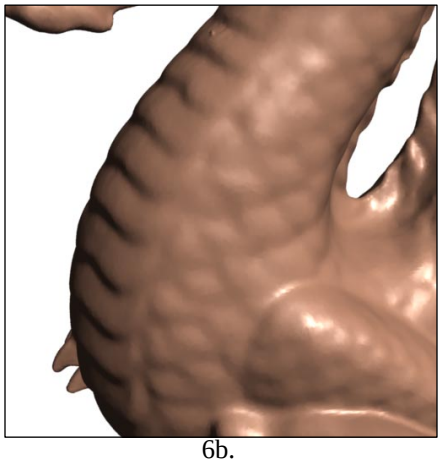
5b.



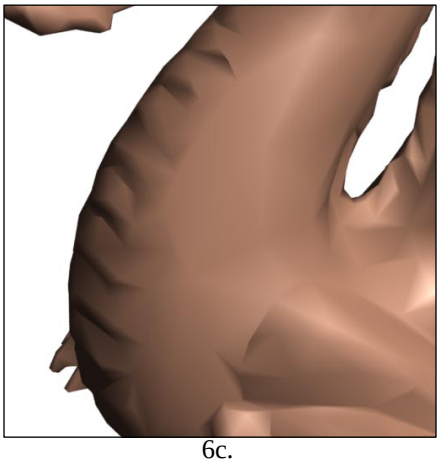
5c.



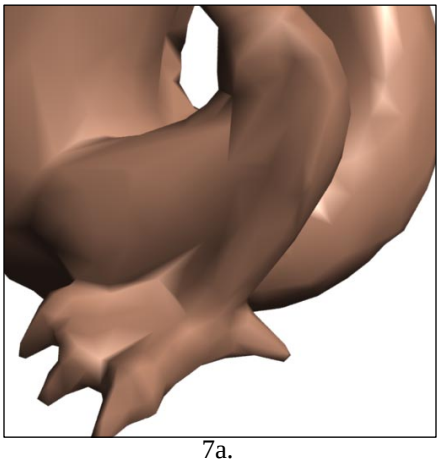
6a.



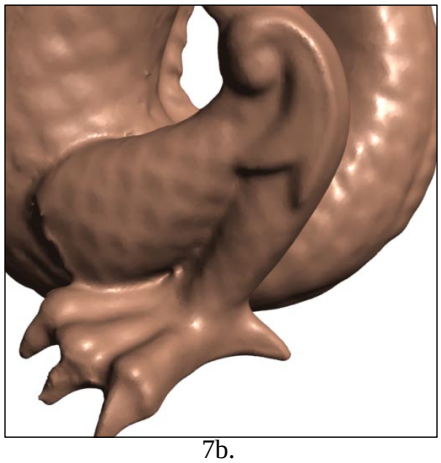
6b.



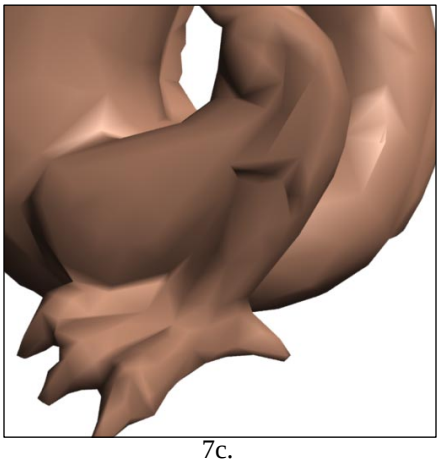
6c.



7a. Model optimized for flat shading.



7b. Original model.



7c. Model optimized for Gouraud shading.

8a. Optimized for flat shading.
T = 16,384. time = 2:28:46.
d_{RMS} = 1.63.

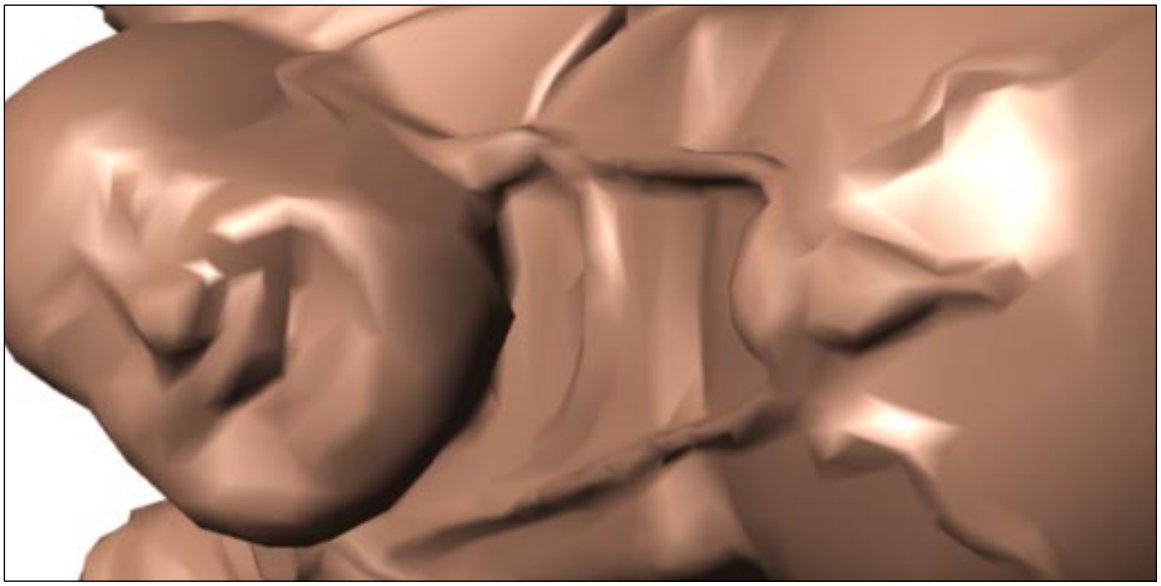
8b. Flat shaded image of original buddha model.
T = 1,087,716.

8c. Optimized for Gouraud shading.
T = 16,384. time = 2:13:08.
d_{RMS} = 2.02.

8d. Optimized for flat shading.
d_{RMS} = 2.26..

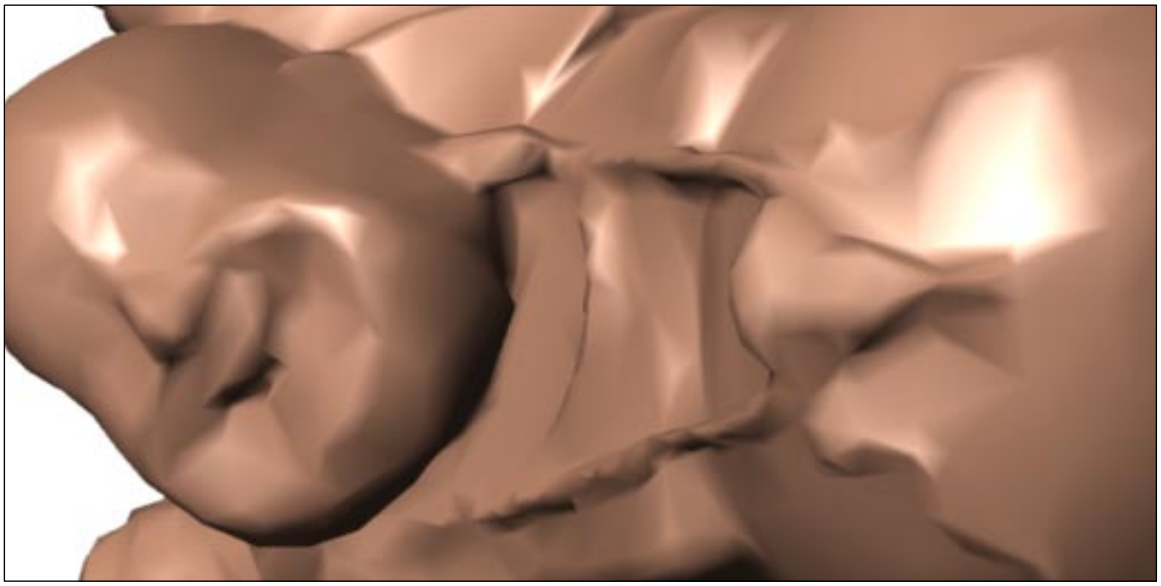
8e. Gouraud shaded image of original buddha model.

8f. Optimized for Gouraud shading.
d_{RMS} = 1.75.

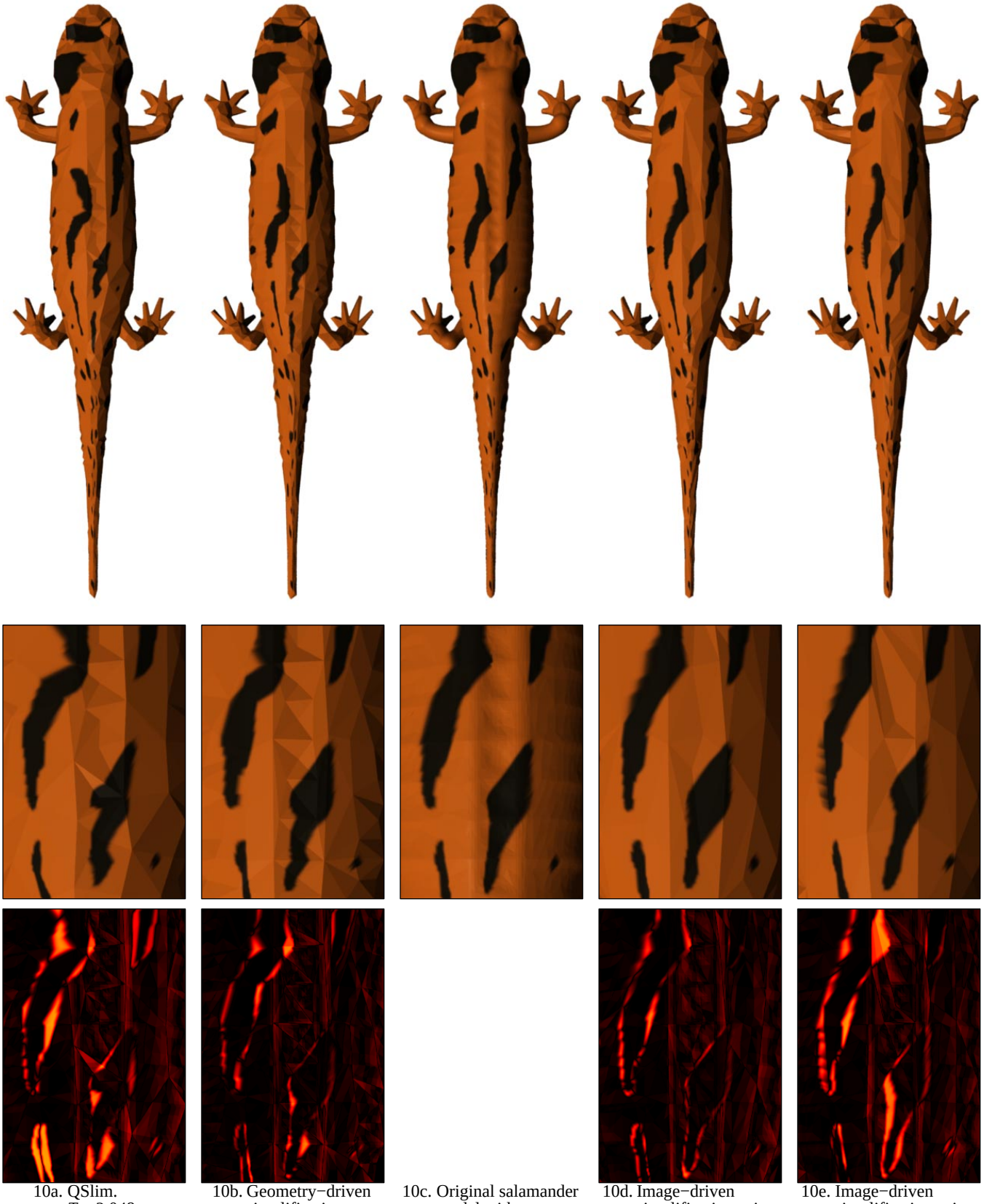


9c. Closeup of Gouraud shaded model optimized for Gouraud shading.

9b. Closeup of Gouraud shaded original model.



9a. Closeup of Gouraud shaded model optimized for flat shading.



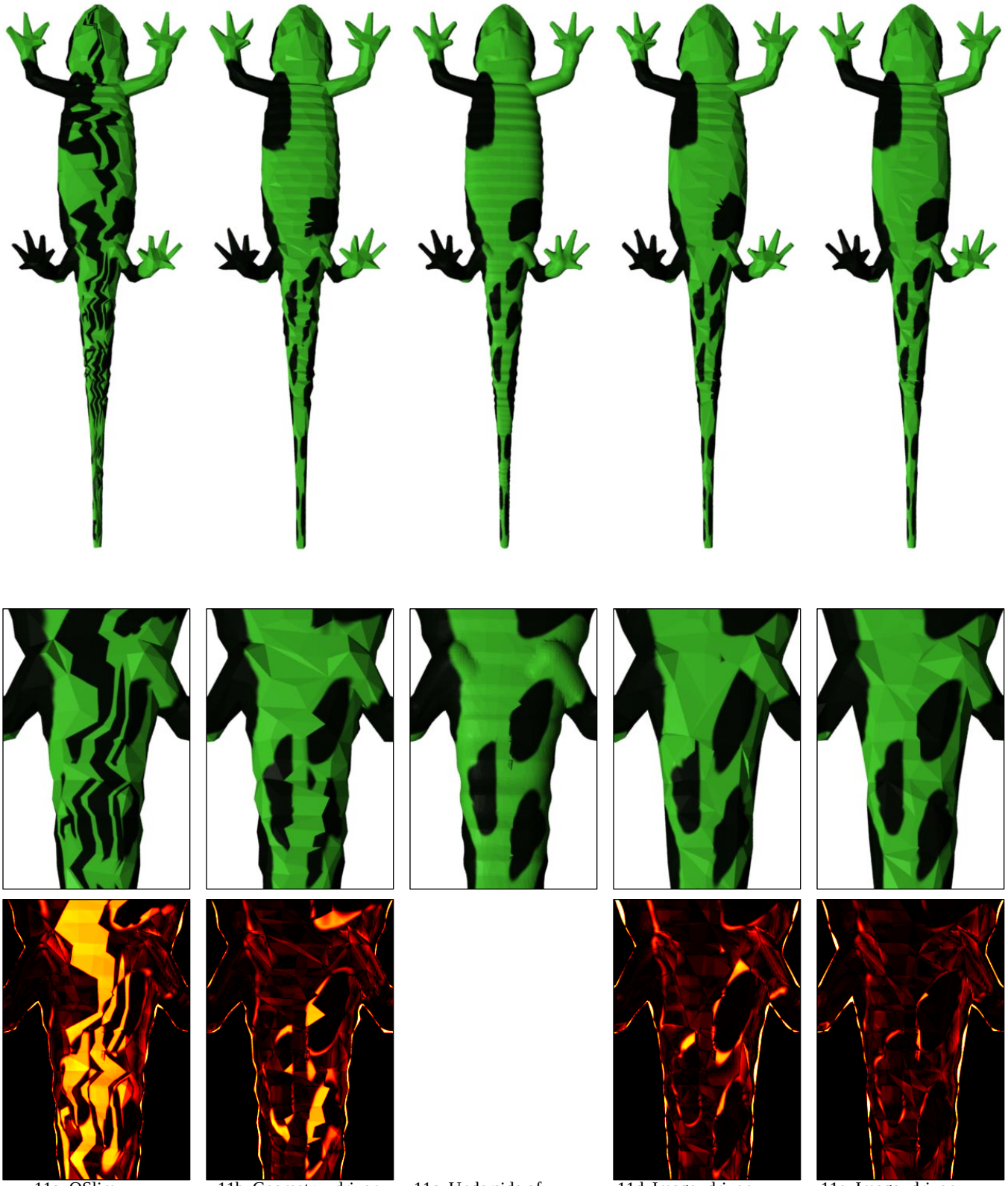
10a. QSlim.
T = 2,048.
time = 0:28.
d_{RMS} = 1.29.

10b. Geometry−driven simplification.
T = 2,047.
time = 1:21.
d_{RMS} = 0.96.

10c. Original salamander model with orange texture.
T = 71,312.

10d. Image−driven simplification using orange texture.
T = 2,047.
time = 17:13.
d_{RMS} = 0.92.

10e. Image−driven simplification using green texture.
T = 2,047.
time = 17:30.
d_{RMS} = 1.00.



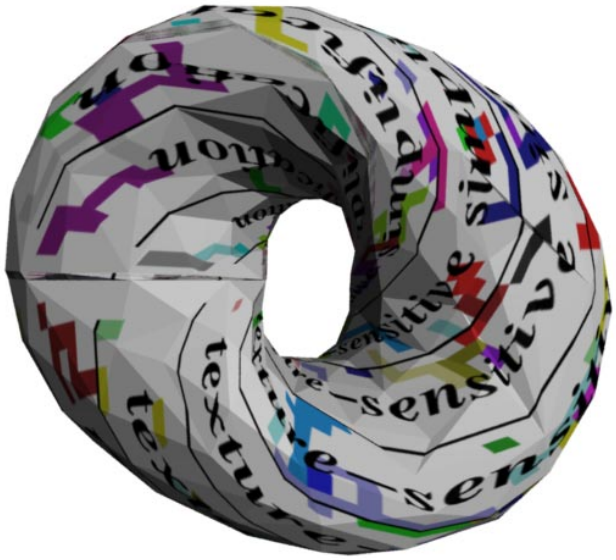
11a. QSlim.
T = 2,048.
time = 0:28.
d_{RMS} = 2.48.

11b. Geometry−driven simplification.
T = 2,047.
time = 1:21.
d_{RMS} = 1.18.

11c. Underside of original salamander model with green texture.
T = 71,312.

11d. Image−driven simplification using orange texture.
T = 2,047.
time = 17:13.
d_{RMS} = 1.10.

11e. Image−driven simplification using green texture.
T = 2,047.
time = 17:30.
d_{RMS} = 0.90.

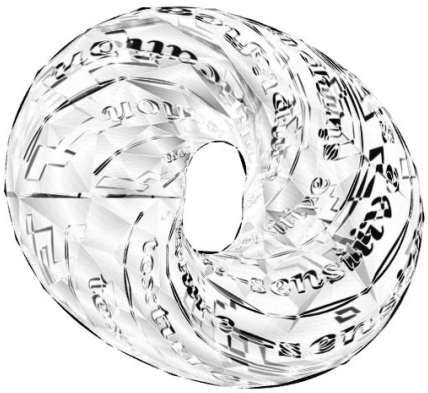


12a. QSlim. T = 512. time = 1:04.

12b. Geometry−driven simplification. T = 512. time = 1:03.

12c. Image−driven simplification. T = 512. time = 5:33.

12d. Original torus model. T = 65,536.



13a. Difference image for QSlim. d_{RMS} = 8.76.

13b. Difference image for geometry−driven simplification. d_{RMS} = 7.29.

13c. Difference image for image−driven simplification. d_{RMS} = 5.99.