Feature-Based Surface Parameterization and Texture Mapping

EUGENE ZHANG, KONSTANTIN MISCHAIKOW, and GREG TURK
Georgia Institute of Technology

Surface parameterization is necessary for many graphics tasks: texture-preserving simplification, remeshing, surface painting, and precomputation of solid textures. The stretch caused by a given parameterization determines the sampling rate on the surface. In this article, we present an automatic parameterization method for segmenting a surface into patches that are then flattened with little stretch.

Many objects consist of regions of relatively simple shapes, each of which has a natural parameterization. Based on this observation, we describe a three-stage feature-based patch creation method for manifold surfaces. The first two stages, genus reduction and feature identification, are performed with the help of distance-based surface functions. In the last stage, we create one or two patches for each feature region based on a covariance matrix of the feature's surface points.

To reduce stretch during patch unfolding, we notice that stretch is a 2×2 tensor, which in ideal situations is the identity. Therefore, we use the *Green-Lagrange tensor* to measure and to guide the optimization process. Furthermore, we allow the boundary vertices of a patch to be optimized by adding *scaffold triangles*. We demonstrate our feature-based patch creation and patch unfolding methods for several textured models.

Finally, to evaluate the quality of a given parameterization, we describe an image-based error measure that takes into account stretch, seams, smoothness, packing efficiency, and surface visibility.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric algorithms, languages, and systems*

General Terms: Algorithms

Additional Key Words and Phrases: Surface parameterization, segmentation, texture mapping, topology

1. INTRODUCTION

Surface parameterization is a well-studied problem in computer graphics. In general, surface parameterization refers to segmenting a 3D surface into one or more patches and unfolding them onto a plane without any overlap. Borrowing terminology from mathematics, this is often referred to as creating an *atlas* of *charts* for a given surface. Surface parameterization is necessary for many graphics applications in which properties of a 3D surface (colors, normal) are sampled and stored in a texture map. The quality of the parameterization greatly affects the quality of subsequent applications. One of the most

This work is supported by NSF grants ACI-0083836, DMS-0138420, and DMS-0107396.

Authors' addresses: E. Zhang, School of Electrical Engineering and Computer Science, Oregon State University, 102 Dearborn Hall, Corvallis, OR 97331-3202; email: Zhange@cc.gatech.edu; K. Mischaikow, Center for Dynamical Systems and Nonlinear Studies, School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332; email: mischaik@math.gatech.edu; G. Turk, College of Computing/GVU Center, Georgia Institute of Technology, Atlanta, GA 30332; email:turk@cc.gatech.edu.

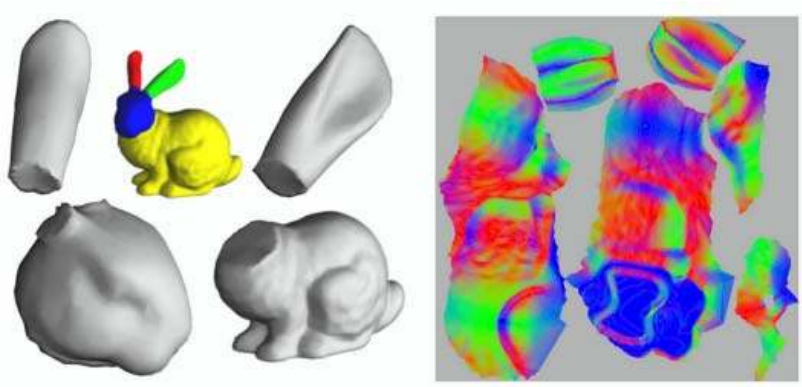

Fig. 1. The feature regions (left) and the unfolded patches (right, colors are used to encode surface normal) for the bunny surface using our algorithm.

important quality measurements is stretch. When unfolding a surface onto a plane, stretching occurs if the surface contains highly spherical or hyperbolic regions. High stretch in a parameterization results in an uneven sampling rate across the surface.

We observe that many objects can be decomposed into a set of "simple" shapes that roughly approximate cylinders, cones, flat disks, and spheres. Cylinders, cones, and planes are developable surfaces, which are Euclidean by nature. Unfolding them results in little stretch, without any overlap. In this article, we make use of some distance-based surface functions to divide a manifold surface into feature regions, each of which is similar to one of the simple shapes. In Figure 1 (left), the bunny surface is decomposed into four feature regions (ears, head, and body) using our segmentation algorithm. These regions are converted into patches and unfolded with little stretch (right, colors are used to encode surface normal).

Existing patch unfolding techniques are often carried out in two stages: an initial patch layout to achieve some objective such as conformal mapping, followed by an interior vertex optimization based on some stretch metric. We observe that an ideal surface parameterization between a patch and its textural image is an isometry, that is, a bijective map that preserves distances. The Green-Lagrange deformation tensor has the property that it measures anisotropic stretch faithfully and penalizes undersampling more severely than oversampling. In addition, it can be seen as a balance between area-preserving mappings and conformal mappings. We use this metric to guide the vertex optimization process for patch unfolding. In addition, we use what we call *scaffold triangles* to convert the original boundary vertices into "interior" vertices, which can then be freely moved around within the same optimization framework. This is a new way of creating nonconvex patches that may even have holes.

In this article, we present an automatic surface parameterization technique that consists of several new ideas and improves upon existing techniques in the following aspects. For patch creation, instead of relying on local curvature information for feature detection as in the case of most previous parameterization methods, we extract and segment large protrusions based on the topological analysis of some distance-based surface functions that are global in nature. This results in a small number of large patches that can be unfolded with relatively little stretch. For patch unfolding, we use the Green-Lagrange tensor to measure stretch and to guide the stretch optimization process. In addition, we create a "virtual boundary" to allow the patch boundaries to be optimized, without the need to check for global self-intersections. Finally, we describe a novel image-based quality metric for surface parameterization that implicitly takes into account stretch, seams, packing efficiency, smoothness, and surface visibility.

The remainder of the article is organized as follows. In Section 2, we review existing surface parameterization techniques. Then, we present our feature-based patch creation method in Section 3, followed by our new balanced stretch metric in Section 4.1, boundary vertex optimization technique in Section 4.2, and our packing algorithm in Section 5. In Section 6, we show the results of applying our technique to various 3D models and describe our image-based quality metric for parameterization techniques. Section 7 provides a summary of our contributions and a discussion of some possible future work.

2. PREVIOUS WORK

There has been a considerable amount of recent work in the graphics community on building a surface parameterization by unfolding a polygonal surface into planar patches. Much of the motivation for this is for *texture mapping*, the mapping of pixels from a rectangular domain (the *texture map*) onto a surface that is described by a collection of polygons. The surface parameterization problem is to subdivide the given surface into a (hopefully small) number of patches that are then flattened onto a plane and arranged in a texture map. Uses for surface parameterization include surface painting [Hanrahan and Haeberli 1990], fast rendering of procedural textures [Perlin 1985; Turk 2001; Wei and Levoy 2001; Carr and Hart 2002], applying photographed color variations onto digitized surfaces [Cignoni et al. 1998], and creating normal maps from detailed geometry [Sander et al. 2001]. These same parameterization methods may also be used for *remeshing*, that is, for creating a new mesh from the original surface [Alliez et al. 2002]. Remeshing can be used to improve the triangle shapes, to vary the triangle size according to curvature details, and to induce semi-regular tessellations. Recently, octrees have been used to store colors in 3D for surface texturing without any parameterization [Benson and Davis 2002; DeBry et al. 2002]. Although octree techniques are supported with programmable GPU's, they are not yet directly supported by graphics hardware.

2.1 Patch Creation

There are two common approaches to the patch creation problem. The first of these is to find a single cut for the surface that makes the modified surface topologically equivalent to a disk [Piponi and Borshukov 2000; Gu et al. 2002; Sheffer and Hart 2002; Erickson and Har-Peled 2002; Ni et al. 2004]. This approach has the virtue of creating as few seams as possible, but will often introduce large stretch between the patch and the surface. Such stretching is undesirable because different portions of the surface are represented using quite different amounts of color detail, as measured in pixel resolution in the texture map.

The other major approach is to divide the surface into a collection of patches that can be unfolded with little stretch [Eck et al. 1995; Lee et al. 1998; Sander et al. 2001; Alliez et al. 2002; Lévy et al. 2002; Sorkine et al. 2002]. Though stretch is minimized, this approach creates seams between the patches. These seams cause problems when creating textured images of the surface because the color variation across the seams must be treated with extreme care or the seams will be noticeable. Some methods create small disk-like patches [Eck et al. 1995; Lee et al. 1998; Sander et al. 2001; Alliez et al. 2002], while others attempt to create large patches that match the features contained in the object [Lévy et al. 2002; Sorkine et al. 2002; Katz and Tal 2003]. Our own work takes this latter approach. We cut the surface into multiple patches, but according to the large geometric features of the surface. For example, we would like to recognize the head and limbs of an animal as important features and to create patches that respect these features. The work of Lévy et al. [2002] and Katz and Tal [2003] have similar goals, although their feature-based patch creation methods are quite different than our own.

The definition of the term "geometric feature" varies in different contexts. For surface reconstruction and mesh simplification, features are often defined in terms of local curvature. This is reasonable

because high curvature regions are exactly what these applications are trying to preserve. On the other hand, surface parameterization algorithms incur higher stretch on a smooth surface with long thin protrusions than a noisy surface with small protrusions. In this work, we define geometric features as large protrusions, and our algorithm segments a surface based on its features by performing topological analysis of some distance-based surface functions.

2.2 Patch Unfolding

There have been many patch unfolding techniques. The classical approach treats the patch unfolding problem as finding the minimum of some functional that measures the difference between a parameterization with isometry [Eck et al. 1995; Floater 1997]. First, the boundary vertices are assigned initial positions (usually on a circle or a square). Then the parameterization for the interior vertices is determined by solving a large linear system or through a nonlinear optimization process. Others have used stretch measures such as the Green-Lagrange deformation tensor [Maillot et al. 1993] and a variant of Dirichlet energy [Hormann and Greiner 1999]. Sander et al. [2001] define a geometric stretch metric that is based on the average and maximal stretch in all directions of a triangle. Sorkine et al. [2002] and Khodakovsky et al. [2003] have devised stretch metrics based on the maximum and minimum eigenvalues of the stretch tensor. Sander et al. [2001] also propose a post-processing vertex optimization step that improves their geometric stretch. As we describe later, Sander's patch optimization approach was an inspiration for our own work. To allow the boundary vertices of a patch to be free from the arbitrary initial assignment, Lévy et al. [2002] use a least-squares conformal mapping, and Desbrun et al. [2002] propose an equivalent formulation, *Discrete, Natural Conformal Parameterization*. Sander et al. [2002] allow boundary vertices to move, while checking for global intersections. Lee et al. [2002] add layers of "virtual boundaries" as part of the edge springs to allow the patch boundaries to have natural shapes. Recently, Sheffer and deSturler [2001; 2002] propose to use an *angle-based flattening* approach for patch unfolding. This approach measures stretch in term of the angles deficits between the triangles on the surface and their textural images, and it removes the need to check for global self-intersections.

3. FEATURE-BASED PATCH CREATION

Our feature-based patch creation method is carried out in three stages:

(1) *genus reduction*: a surface with handles (nonzero genus) is converted into a genus zero surface.
(2) *feature identification*: a genus zero surface is divided into a number of relatively simple shapes.
(3) *patch creation*: every simple shape is cut into one or two topological disks.

For both genus reduction and feature identification, we build a surface-based Reeb graph based on the *average geodesic distance* introduced by Hilaga et al. [2001]. This graph consists of vertices and edges in the mesh surface, and we call it an *embedded Reeb graph*. When properly constructed, this graph reveals the location of the handles and protrusions in the surface. Since our goal is to create patches that are topological disks, we need to perform our operations in a topologically consistent manner. For this purpose, we use surface region growing for all three stages: genus reduction, feature identification and patch creation. Starting from an initial triangle, we grow a region by adding one triangle at a time until the whole surface has been covered or until some other stopping criterion has been met. We will now describe the average geodesic distance function and the embedded Reeb graph that it induces.

3.1 The Average Geodesic Distance Function

The *average geodesic distance* function was introduced by Hilaga et al. [2001] for the purpose of shape matching. This is a function $A(\mathbf{p})$ that takes on a scalar value at each point $\mathbf{p}$ on the surface S. Let $g(\mathbf{p}, \mathbf{q})$ be the geodesic distance between two points $\mathbf{p}$ and $\mathbf{q}$ on S. Then the average geodesic distance

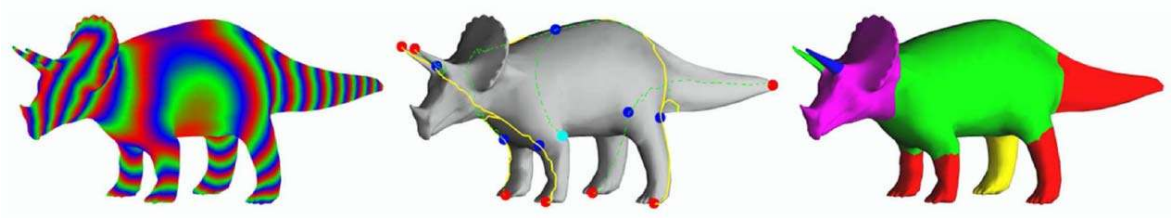

Fig. 2. The *average geodesic distance* function (AGD) on the dinosaur model is color-coded in the left of this figure. The global minimum is located underneath the belly, colored in red. Levelsets are painted in repeated patterns of red, green, and blue. Notice that the tips of large protrusions (horns, legs, tail) are local maxima of AGD. The middle, figure shows the *embedded Reeb graph* created by surface-growing based on AGD. Local maxima are indicated by red spheres, and saddle points are highlighted by blue spheres. Successive *critical points* are connected by surface paths shown in solid yellow (visible) and dash green (hidden). The final surface segmentation result based on our algorithm is shown on the right.

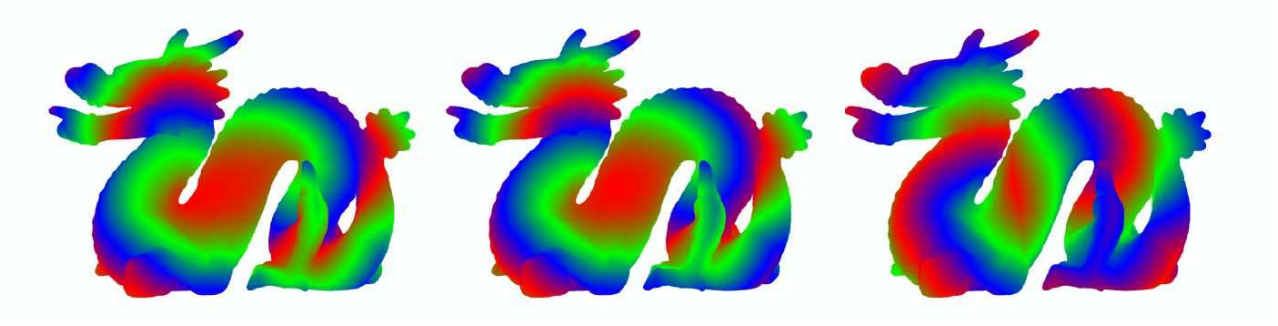

Fig. 3. Comparison among three AGD functions for the dragon: AGD_1 (left), AGD_2 (middle), and AGD_∞ (right).

of $\mathbf{p}$ is defined as follows:

$$A(\mathbf{p}) = \frac{\int_{\mathbf{q}\in S} g(\mathbf{p},\mathbf{q}) d\,\mathbf{q}}{Area(S)}. \tag{1}$$

$A(\mathbf{p})$ is a member of the following set of functions:

$$A_n(\mathbf{p}) = \sqrt[n]{\frac{\int_{\mathbf{q}\in S} g^n(\mathbf{p},\mathbf{q}) d\,\mathbf{q}}{Area(S)}} \tag{2}$$

with $n = 1$. When $n \to \infty$, $A_\infty(\mathbf{p}) := \lim_{n\to\infty} A_n(\mathbf{p}) = \max_{\mathbf{q}\in S} g(\mathbf{p},\mathbf{q})$, which measures the maximal distance between $\mathbf{p}$ and any point on S. We define for $n = 1, 2, \ldots, \infty$

$$AGD_n(\mathbf{p}) := \frac{A_n(\mathbf{p})}{\min_{\mathbf{q}\in S} A_n(\mathbf{q})}. \tag{3}$$

For any $n \geq 1$, AGD_n has several useful properties. First, its value measures how "isolated" a point is from the rest of the surface. Second, its local maxima coincide with the tips of the geometric features contained in the model. Third, it is scale-invariant and can be used to compare features from different shapes. Figure 2 (left) shows a polygonal model of a dinosaur, color-coded according to AGD_2. The red region on the dinosaur's belly signifies that points in this region have low values of AGD_2. Higher values adjacent to this middle region are colored in green and then in blue. The colors then cycle repeatedly through red, green, and blue. Note that the tips of the large features of this object (legs, horns, tail) are marked by local maxima of AGD_2. (In subsequent sections we will use the term *local maxima of AGD* and *tips* interchangeably.) In practice, we use AGD_2 since it seems to produce smooth results. Figure 3

compares the levelsets of the following three functions on the dragon: AGD_1 (left), AGD_2 (middle), and AGD_∞ (right). From now on, we will use the term *AGD* to mean AGD_2.

For a genus zero surface, we use AGD to identify and measure its geometric features. The tip of a protrusion is a local maximum. Larger values at local maxima signify larger protrusions. Notice that a noisy surface often contains many small bumps that correspond to local maxima with relatively small AGD values. Creating patches based on these bumps will increase the amount of seams, without significantly reducing stretch. Therefore, we only consider local maxima whose AGD values are above a threshold. In practice, we find choosing any number in [1.3, 1.5] as the threshold for minimal feature size produces reasonable results, and we use 1.4 for all our test models.

Computing AGD exactly would be quite costly. We closely follow the algorithm of Hilaga et al. [2001] to quickly compute a satisfactory approximation of AGD. Briefly, the geodesic distances are not calculated from all the surface points, but rather from a small number of evenly spaced points on the surface. We find the geodesic distances from each of these points to all other points efficiently using the fast-marching method for surfaces [Kimmel and Sethian 1998].

3.2 Building an Embedded Reeb Graph

To find handles and large protrusions in a model, we perform topological analysis of AGD and construct an embedded Reeb graph Γ that is induced by AGD. The leaf nodes of Γ are situated at the tips of protrusions, and the loops in Γ reveal the existence of handles. We construct Γ by performing region-growing in the increasing order of AGD and tracking the topological changes in the wavefront. This is based upon ideas from *Morse Theory* [Milnor 1963] and *Reeb graphs* [Reeb 1946], which we will review here.

Let f be a smooth function defined on a smooth surface $S \subset \mathbb{R}^3$. For any point $\mathbf{p}_0 \in S$, let $\mu = (u, v)$ be a parameterization of some neighborhood of $\mathbf{p}_0$ in S such that $\mu(0, 0) = \mathbf{p}_0$. The *gradient* ∇f and the *Hessian* Hf are defined as follows:

$$\nabla f = \begin{pmatrix} \partial f/\partial u \\ \\ \partial f/\partial v \end{pmatrix}, \quad Hf = \begin{pmatrix} \partial^2 f/\partial u^2 & \partial^2 f/\partial u \partial v \\ \\ \partial^2 f/\partial u \partial v & \partial^2 f/\partial v^2 \end{pmatrix}. \tag{4}$$

$\mathbf{p}_0$ is a *critical point* of f if $\nabla f(0, 0) = 0$. Otherwise, $\mathbf{p}_0$ is *regular*. A critical point $\mathbf{p}_0$ is said to be *nondegenerate* if $Hf(0, 0)$ does not have any zero eigenvalues. In this case, $\mathbf{p}_0$ can be classified as a mimimum/saddle/maximum if $Hf(0, 0)$ has zero/one/two negative eigenvalues. f is *Morse* over S if f possesses no degenerate critical points. Morse theory relates the critical points of Morse functions to the topology of the underlying surface. For instance, when S is a closed orientable 2-manifold with *Euler characteristic* $\chi(S)$ (twice the number of handles plus two), the following is true for any Morse function f defined on S with α maxima, β saddles, and γ minima.

$$\alpha - \beta + \gamma = \chi(S). \tag{5}$$

Banchoff extends Morse theory to triangular meshes [1970].

A continuous function $f : S \to \mathbb{R}$ induces a Reeb graph Γ_f, which can be used to reveal the topological and skeletal structures of S [Hilaga et al. 2001]. Formally, we define an equivalence relationship $\sim_f$ on S as follows. Let $\mathbf{p}, \mathbf{q} \in S$ be two points, then $\mathbf{p} \sim_f \mathbf{q}$ if and only if $f(\mathbf{p}) = f(\mathbf{q})$ and $\mathbf{p}$ and $\mathbf{q}$ belong to the same connected component of $f^{-1}(f(\mathbf{p}))$. Notice f does not have to be Morse. Figure 4 illustrates an example Reeb graph that corresponds to the vertical height function, defined on a 3D surface. Many applications make use of Reeb graphs, such as shape-matching [Hilaga et al. 2001] and topological simplification [Wood et al. 2004]. AGD is, in general, not Morse. For instance, it is a constant function on a sphere, in which case every point is a degenerate critical point. Axen and Edelsbrunner [1998] show that a function can be perturbed into a Morse function with surface wave traversal, provided

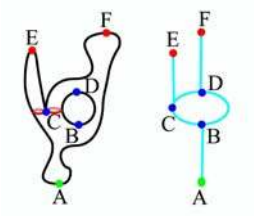

Fig. 4. An example of a Reeb graph (right) for a vertical height function, defined on a 3D surface of genus one (left). The critical points are highlighted by colored spheres (red for maxima, green for minima, and blue for saddles). The number of loops in the graph equals the number of handles in the surface.

that the mesh is properly subdivided. We use a similar strategy except that we record critical triangles instead of critical vertices.

Our algorithm for building an embedded Reeb graph Γ starts with computing AGD for every vertex. For a triangle $T = \{v_1, v_2, v_3\}$, we define $AGD(T) = \min\{AGD(v_1), AGD(v_2), AGD(v_3)\}$. Starting with a triangle whose AGD value equals the global minimum, we add one triangle at a time in the increasing order of the AGD until the surface is covered. The boundary of the visited region consists of a number of loops. We label a triangle, when it is added, according to one of the following five criteria:

(1) Minimum: where one new boundary loop starts. For our application, there is only one such triangle, one of the global minima of the AGD.
(2) Maximum: where one boundary loop vanishes. This is the tip of a protrusion.
(3) Splitting saddle: where one boundary loop intersects itself and splits into two.
(4) Merging saddle: where two boundary loops intersect and merge into one. This signifies the formation of a handle.
(5) Regular: where the number of boundary loops does not change.

A triangle that is not regular is a critical triangle. Let n be the genus of the surface, and let N_{max}, N_{min}, N_{ss}, and N_{ms} be the number of the triangles that are maxima, minima, splitting saddles, and merging saddles. Then we have,

$$N_{ms} = n \tag{6}$$

$$N_{max} - N_{ss} + N_{ms} + N_{min} = 2. \tag{7}$$

Equation 7 corresponds to the handle-body decomposition of a closed and orientable piecewise linear 2-manifold [Rourke and Sanderson 1972]. Interested readers may refer to Lopes et al. [2003] for more details. For our application, $N_{min} = 1$. Furthermore, we mark the center of a critical triangle as the position of the corresponding critical point. The region on the surface swept out between a pair of critical triangles (not including these critical triangles) is homeomorphic to a cylinder without caps. Let A and B be a pair of critical triangles, and assume that A is visited earlier than B. We refer to A as the *parent critical triangle* and B as the *child critical triangle*. For a genus zero surface, every child critical triangle has a single parent. For surfaces with a genus greater than zero, a child critical triangle may have one or two parents. Let R_{AB} be the connecting region between A and B, which consists of a set of regular triangles $= \{T_1, \ldots, T_k\}$ in the order of which they are visited. There is a shortest path that connects A and B using the edges of the set of triangles $\{A\} \bigcup \{B\} \bigcup R_{AB}$. We construct the embedded Reeb graph by finding the shortest paths between every pair of parent/child critical triangles.

As mentioned earlier, the embedded Reeb graph Γ is much like a Reeb graph that corresponds to *AGD*. It reveals the distribution of the geometric features over the surface. The middle of Figure 2 shows the embedded Reeb graph of the dinosaur. Local maxima are highlighted with red spheres, while

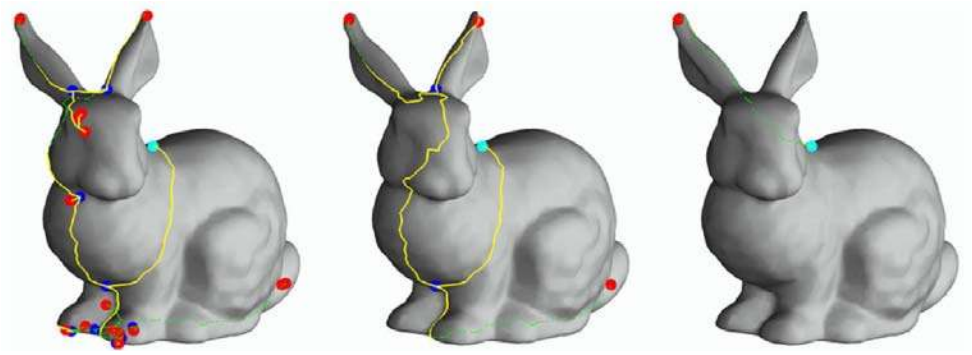

Fig. 5. Embedded Reeb graphs for the bunny surface with different filtering constants α: 1.01 (left), 1.1 (middle), and 1.5 (right). We use 1.1 as the filtering constant for all the test models.

blue spheres indicate the location of splitting saddle points. The global minimum is marked with a light blue sphere on the belly. Successive critical points are connected by paths on the surface, which are drawn in solid yellow (visible) and dash green (hidden). Note that local maxima coincide with the tips of the geometric features (horns, feet, and tail).

Since complex surfaces often contain many small protrusions (bumps), the embedded Reeb graph can contain an excessive number of local maxima and saddle points. This increases the subsequent processing time since the number of features is much more than what we consider large (or "persistent" as described in Edelsbrunner et al. [2003]). We use the following filtering scheme to weed out extra local maxima and splitting saddle points. During surface-growing, we alter the order in which triangles are added. To be more specific, let $\mathbf{t}$ be the unvisited triangle with the smallest AGD value. If adding $\mathbf{t}$ causes a boundary to split, we look for other triangles that could be added without causing a boundary split. If one of these triangles, $\mathbf{t}'$ satisfies:

$$AGD(\mathbf{t}') < \alpha AGD(\mathbf{t}) \tag{8}$$

where α is a global filtering constant, then we add $\mathbf{t}'$ instead of $\mathbf{t}$. When there are multiple choices, we choose the triangle with the smallest AGD value. Our filtering process is related to the concept of *topological persistence and simplification* [Edelsbrunner et al. 2003], but with a different scalar function and a different measure for persistence. Also, the simplification process is implicit. We apply the filtering scheme to the bunny surface (Figure 5) with three different α's: 1.01 (left), 1.1 (middle), 1.5 (right). Notice the excessive saddle points and maxima appear in the head and the paws when $\alpha = 1.01$ (left). When $\alpha = 1.1$ (middle), the local maxima that reveal large geometric structures are kept (the tips of ears and the center of tail). Excessive filtering may result in a trivial embedded Reeb graph, for example, $\alpha = 1.5$ (right). This becomes a classical tradeoff between denoising and overblurring. In practice, we find $\alpha \in [1.1, 1.3]$ works well, and we use $\alpha = 1.1$ for all the test models shown in the article.

For a genus $n > 0$ surface, there are n loops in the embedded Reeb graph Γ that are homologically inequivalent and form the bases of all loops in Γ. In Section 3.4, we describe how we use these loops for genus reduction, that is, converting a genus n surface into a genus zero surface.

3.3 Feature Identification

Once the tip of a protrusion is located, we construct a closed curve γ on the surface that separates the feature from the remaining body. Using the terminology from Erickson and Har-Peled [2002], γ is a *separating cycle*. We compute γ in two steps. First, we find a separating region R corresponding to the tip of the protrusion. Next, we construct γ from R.

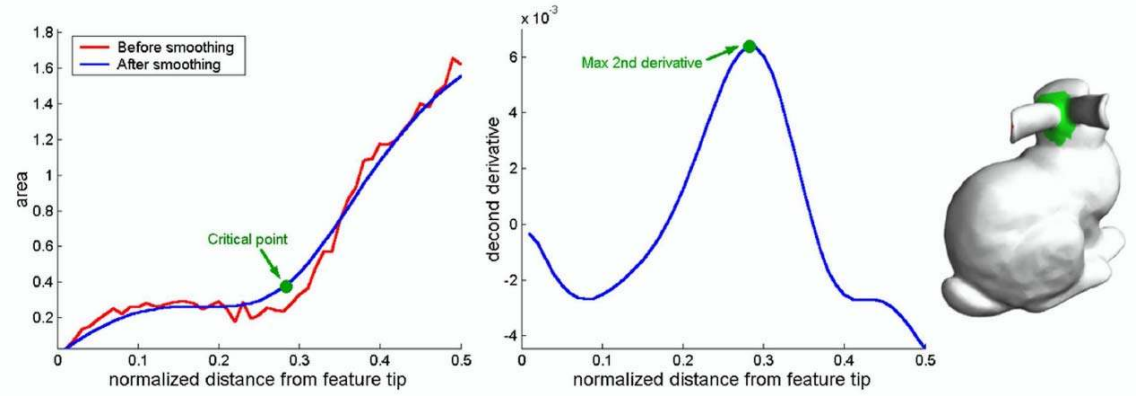

Fig. 6. In this figure, we find a separating region for the bunny's ear. In the left we graph $A(x)$, the area of the regions given by evenly spaced distances from the ear's tip (red), and the smoothed areas (blue). We then calculate $A''(x)$, the second derivative (middle). The maximum of $A''(x)$ corresponds to the place where the ear joins the head (right, the green band).

To find a separating region for the tip point $\mathbf{p}$ of a feature, we first calculate the function $f_{\mathbf{p}}(\mathbf{q}) = g(\mathbf{p}, \mathbf{q})$, the surface geodesic distance function with respect to $\mathbf{p}$. $f_{\mathbf{p}}$ is normalized to take on values in $[0, 1]$. We consider the regions bounded by iso-value curves of this function. Specifically, we divide the interval $[0, 1]$ into k equal sections. Next, by performing region-growing from $\mathbf{p}$, we partition the surface into levelset bands based on the values of $f_{\mathbf{p}}$ in these intervals:

$$M_i := \left\{ \mathbf{q} \in S | \frac{i-1}{k} \leq f_{\mathbf{p}}(\mathbf{q}) \leq \frac{i}{k} \right\} \tag{9}$$

$$A_i := Area(M_i). \tag{10}$$

The construction of levelsets in Equation 9 is inspired by Morse theory. The area of this sequence of bands changes slowly along a protrusion, but it changes abruptly where the feature joins to the rest of the surface. We find the separating region by analyzing $\{A_i\}$, which we treat as a continuous function $A(x)$. Along a perfect cylindrical feature, $A(x)$ is constant. In the case of a cone, the function grows linearly. At places where a protrusion joins the main body, $A(x)$ will have a sudden increase, and this will be the boundary of the feature. We find these increases by looking for the maxima in $A''(x)$, the second derivative of $A(x)$. To eliminate small undulations in $A(x)$, we first low-pass filter $A(x)$ using a Gaussian function for N times. Both k and N affect the efficiency of separating region detection. The larger k is, the more samples are used to discretize $A(x)$, and the more likely small noise will be considered as potential places for the separating region. Similarly, if N is too large, the location for the separating region may be shifted or even lost. In practice, we use $k = 100$ and $N = 30$ for all our test models. These choices seem to produce reasonable results. Figure 6 illustrates the process of feature identification for the bunny's ear.

Let m be the location where $A''(m)$ takes on its maximum value. We define the separating region $R := \{\mathbf{q} \in S \mid m - \epsilon \leq f_{\mathbf{p}}(\mathbf{q}) \leq m + \epsilon\}$, where we typically use $\epsilon = 0.02$. By choosing a positive ϵ, we intentionally make R large for two reasons. First, poor triangulations may cause R to be nonseparating, that is, it does not contain a separating cycle. Second, we would like more flexibility to allow the separating cycle (within this region) to be short and smooth.

The topology of the separating region R can be rather complex if there are other features that join the surface in nearby places. The only guarantee R provides is that it indeed separates the feature

from the rest of the surface. We produce a separating cycle γ from R as follows. First, we reduce R into its skeleton, that is, a collection of edges in the surface that separate the feature from the rest of the surface. Dangling edges are removed as well. Gu et al. [2002] perform a similar operation to produce geometry images from meshes. Next, we find a separating cycle ρ from this skeleton. Finally, we construct another separating cycle γ that is based on ρ, but that is in general shorter and smoother. These operations are easy to implement on meshes and we describe them in detail next.

(1) Reduce a separating region R into its skeleton and remove dangling edges. This is achieved by treating R as a 2-complex (with boundary edges) and repeatedly performing "elementary collapses" [Kaczynski et al. 2004], in which a triangle with at least one boundary edge is removed from the complex along with one of the boundary edges. In the end, all 2-cells (triangles) are removed, and the 2-complex is reduced to a 1-complex. When there are multiple choices of boundary edges to collapse, we select the edge with the largest AGD value, which tends to be closer to the feature tip $\mathbf{p}$ than the other edges in the 2-complex. The resulting graph is a skeleton of R with dangling edges. We remove the dangling edges through elementary collapses on the 1-complex. This results in a collection of loops, one of which meets our requirement as the separating cycle. The others fall into two categories: separating cycles for some geometric features inside the feature region, and separating cycles for some geometric features outside the feature region.

(2) Eliminate from the 1-complex separating cycles that are either inside or outside the feature region. To remove the loops outside the feature region, we perform region-growing from the feature tip $\mathbf{p}$ with the constraint that no triangles can be added that cross an edge in the loops computed from the steps in (1). This makes the loops outside the feature region unreachable from $\mathbf{p}$. For loops inside the feature region, the average AGD values of their vertices are, in general, greater than that on the separating cycle. Therefore, these loops can be easily identified and discarded. This step produces a separating cycle ρ.

(3) Shorten and smooth the separating cycle ρ. We choose two vertices $\mathbf{t}_1$ and $\mathbf{t}_2$ on ρ that are the closest to the feature tip $\mathbf{p}$. We find two paths that connect $\mathbf{t}_1$ and $\mathbf{t}_2$ to $\mathbf{p}$, respectively. The two paths divide the feature region into two disjoint regions. Within each region, there is a shortest path between $\mathbf{t}_1$ and $\mathbf{t}_2$. Together, they form a separating cycle, which tends to be shorter and smoother than ρ. By repeating this process twice, we obtain a desired separating cycle γ.

Figure 7 illustrates this process. For a feature point $\mathbf{p}$ and a separating region R (a, the shaded region), we reduce R to its skeleton through elementary collapses (b). Next, loops that are either inside or outside the feature region of $\mathbf{p}$ are eliminated (c). In the bottom row, the separating cycle ρ is shortened and smoothed to produce γ, through step 3 (d-f).

A separating cycle divides S into two surfaces with boundaries. We eliminate these boundaries by "filling in" the holes with triangles. Basically, we compute c, the average position of the boundary vertices, and make c a new vertex. We then triangulate the hole by connecting c to the vertices on the boundary. The filler triangles are subdivided twice, followed by Laplacian smoothing on the newly created vertices. Some filler triangles can be seen where the head has been separated from the neck of the bunny in Figure 1. These filler triangles are flagged so that they have minimal effect on patch unfolding. They become what we call *scaffold triangles*, to be described later.

We repeat the feature identification process for the resulting surfaces until the original surface is divided into a set of feature regions and there are no more feature regions to be found. Figure 1 shows the result of this process on the bunny, in which four regions were created.

Our feature identification algorithm assumes that a single loop always divides the surface into two disjoint regions, which is not necessarily true for surfaces with handles. For these surfaces, the topology of the separating region can be arbitrarily complex where many small handles are clustered together. To avoid dealing with this situation, we perform genus reduction before feature identification. Genus reduction converts a genus $n > 0$ surface into a genus zero surface.

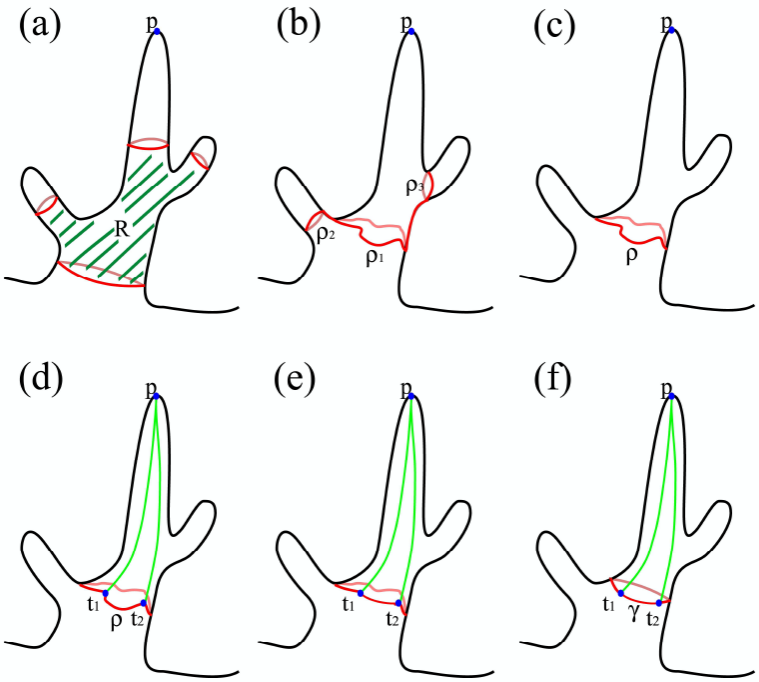

Fig. 7. This figure illustrates our algorithm for producing a separating cycle from a separating region. In (a), a separating region R (shaded) for $\mathbf{p}$ is bounded by the red-curves. Next in (b), R is reduced to its skeleton through elementary collapses. The skeleton consists of three loops: ρ_1, ρ_2, and ρ_3. Notice that ρ_2 is not reachable from $\mathbf{p}$, and ρ_3 has a higher average AGD value than ρ_1. By eliminating them, we obtain a separating cycle $\rho = \rho_1$ (c). To smooth ρ, we find two points $\mathbf{t}_1$ and $\mathbf{t}_2$ in ρ and the shortest paths that connect $\mathbf{t}_1\mathbf{p}$ and $\mathbf{t}_2\mathbf{p}$ (green curves in (d)). Together the two curves divide the region bounded by ρ into two subregions. Inside each subregion, we construct a shortest path between $\mathbf{t}_1$ and $\mathbf{t}_2$ (the red curves in (e) and (f)). The union of the two paths forms a separating cycle γ that is, in general, shorter and smoother than ρ.

3.4 Genus Reduction

For a genus n surface ($n > 0$), a loop does not always divide the surface into two disjoint connected components. Loops with this property are associated to the elements of the first homology group, which form an Abelian group with $2n$ generators. Using the terminology from [Erickson and Har-Peled 2002], these loops are *nonseparating cycles*. Conceptually, the easiest way to think of how these loops arise is to imagine a hollow handle connected to the rest of the surface; one of the loops cuts across the handle and the other follows the handle. Observe that for the first type of loops there are two "passages" back to the rest of the surface. Our strategy for genus reduction is to identify an appropriate nonseparating cycle for every handle and cut the surface open along the cycles. Each operation converts a handle into one or two protrusions and reduces the genus of the surface by one. We repeat this process until the surface contains no handles. Erickson and Har-Peled [2002] have proved that finding the minimal length cuts needed to turn a surface into a disk is NP-hard, so heuristics are used in practice to find cuts that are short in length. Genus reduction may be performed using a number of already existing techniques, including Guskov and Wood [2001], Lazarus et al. [2001], Erickson and Har-Peled [2002], Gu et al. [2002], Sheffer and Hart [2002], and Wood et al. [2004]. We choose to perform genus reduction using the embedded Reeb graph Γ and the same distance function that we use for feature identification. Figure 8 shows our genus reduction algorithm on the dragon (genus one).

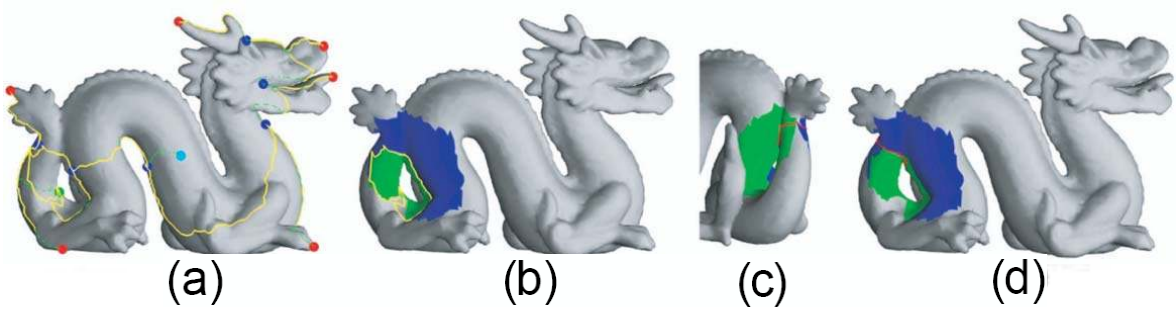

Fig. 8. This figure illustrates the process of genus reduction for the dragon. After the embedded Reeb graph Γ is computed (a, the graph colored in yellow), we find the independent nonseparating cycle contained in Γ (b, yellow) and perform region-growing from this loop in both directions until the wavefronts meet (b, the blue and green regions). The meeting point is shown in (c). We find a path within the green and blue regions that connect the meeting point to the original nonseparating cycle. The paths form a nonseparating cycle (c and d, the red loop), which can be used for turn a handle into one or two protrusions.

First, we compute the embedded Reeb graph Γ induced by AGD (a, the yellow graph) and locate all the basis loops in Γ (b, the yellow loop). Recall (Section 3.2) that a merging saddle point $\mathbf{q}_i$ signals that a handle has formed (a, the green sphere). The start of a handle is a splitting saddle point $\mathbf{p}_i$, which is, in general, located near the ends of the passages that connect the handle to the rest of the surface. We construct a basis loop ρ by computing two shortest paths in Γ that connect $\mathbf{q}_i$ and $\mathbf{p}_i$ (b, yellow loop).

Next, for each basis loop ρ, we create a nearby nonseparating cycle γ for one of the passages by performing region-growing from ρ in the increasing order of the distance function from $\mathbf{p}_i$ (c and d, the red loop). To do so, we treat ρ as the two boundary loops of a region with no interiors. Denote this region R. Since the surface has handles, region-growing from R causes the two boundary loops to meet at a merging saddle point $\mathbf{r}$. Figure 8 (b) shows the shapes of the two regions swept by the two loops when they meet (c, blue and green regions). Within each region, there is a shortest path between $\mathbf{r}$ and ρ. Together, the two paths form a nonseparating cycle γ (d, red loop) that is, in general, shorter and smoother than ρ.

Finally, the surface is cut open along γ and the holes are filled with scaffold triangles. This reduces the genus of the surface by one. We repeat this process until the surface contains no handles, at which point it is ready for feature identification (Section 3.3). Figure 9 shows the nonseparating cycles that are generated using our genus reduction algorithm for three surfaces. Notice that these loops appear in intuitive places and tend to be short, smooth, and nonwinding.

3.5 Patch Creation

Through genus reduction and feature identification, a surface is decomposed into a set of simple shapes (features) that are topological spheres without large protrusions. Our next task is to create one or two patches (topological disks) for every feature shape so that unfolding them results in little stretch. This is carried out in two stages. First, we classify the feature shapes as belonging to one of the following three profiles: a linear ellipsoid, a flat ellipsoid, and a sphere. Next, we create patches for every feature shape based on its profile.

The classification step requires computing the eigenvalues of a covariance matrix for every feature shape F, which is a triangular mesh. We compute the covariance matrix M_F in closed forms by following the method of Gottschalk et al. [1996] that begins by thinking of F as having been sampled "infinitely densely" by points over its surface. First, we calculate the mean μ of these points by integrating over all the triangles. Similarly, we compute M_F relative to μ by performing integrations. The three categories of features are then distinguished based on the eigenvalues from M_F:

Fig. 9. This figure displays the nonseparating cycles that are used for genus reduction for the Buddha (a), the dragon (upper right), and the feline (lower right). Each separating cycle consists of a sequence of edges in the surface that form a closed loop. Notice they appear in intuitive places. Furthermore, they tend to be short, smooth, and nonwinding.

—Three nearly equal eigenvalues (a sphere).

—One eigenvalue much larger than the other two (a linear ellipsoid).

—Two nearly equal eigenvalues that are much larger than the third (a flat ellipsoid).

Let α, β, γ be the three eigenvalues of M_F after normalization such that $\alpha^2 + \beta^2 + \gamma^2 = 1$. The set of all valid configurations is:

$$C := \{(\alpha, \beta, \gamma) | \alpha^2 + \beta^2 + \gamma^2 = 1, \alpha, \beta, \gamma \geq 0\}. \tag{11}$$

C is a spherical triangle in the first octant. There are seven special configurations that correspond to the three linear ellipsoids, the three flat ellipsoids, and the perfect sphere. By building the Voronoi regions on C using spherical distances, we can classify every shape based on the position of its configuration in C. Alternatively, one can use the classification measure proposed by Kindlmann and Weinstein [1999], which produces similar classifications for our test models.

In the case of a linear ellipsoid, we find a pair of points $(\mathbf{p}, \mathbf{q})$ on F such that they achieve the maximum surface distance. This can be approximated by letting $\mathbf{p}$ be a global maximum of AGD, and letting $\mathbf{q}$ be the point that is furthest away from $\mathbf{p}$ on the surface. We then find the shortest path γ between $\mathbf{p}$ and $\mathbf{q}$ and cut the surface along γ by duplicating all of its vertices except $\mathbf{p}$ and $\mathbf{q}$. This converts the surface into a single patch (a topological disk).

For the flat ellipsoid case, we first identify the eigenvector associated with the smallest covariance eigenvalue. Then we find the two most distant surface points x_1 and x_2 along this vector in opposite directions away from the surface's center μ. Using region-growing, we find the Voronoi regions for x_1 and x_2. Both regions are homeomorphic to a disk.

In the case of a sphere, we could treat it as a flat ellipsoid and create two patches that are much like hemispheres. However, unfolding these patches would cause high stretch. Instead, we use an approach that is inspired by the two identical patches of a baseball (see Figure 10, upper left). We construct these regions based on two C-shaped curves, each of which travels halfway around one of the two mutually

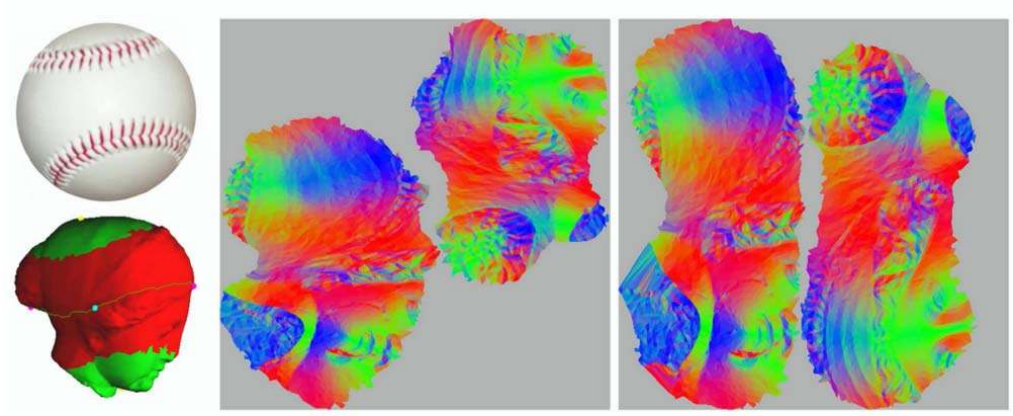

Fig. 10. The "baseball" decomposition of the Venus (lower left) and the corresponding normal maps from patch unfolding using different stretch metrics: Sander's metric [Sander et al. 2001] (middle) and the Green-Lagrange tensor (right, Section 4.1).

perpendicular great circles. To compute these curves, we find the three pairs of antipodal points on the surface that passes through the surface center μ, along the three eigenvector directions. Call these points $x_1, x_2, y_1, y_2, z_1, z_2$. One C-curve passes through x_1, y_1, x_2, and the other connects z_1, y_2, z_2. Using region-growing, we compute the "baseball decomposition" of the surface by building the surface Voronoi regions corresponding to the C-curves. The lower left of Figure 10 shows one of these curves and their corresponding Voronoi regions (red and green) for the Venus. Also shown in the same figure are the normal maps corresponding to the decomposition with two different patch unfolding methods: Sander's metric [Sander et al. 2001] (middle), and the Green-Lagrange tensor (right, Section 4.1). In the next section, we will show that patch unfolding using the Green-Lagrange tensor results in less overall stretch than using Sander's metric.

Sometimes a feature is a curved long cylinder, such as the feline's tail, whose covariance analysis is similar to that a flat ellipsoid or a sphere. In this case, the center μ is situated outside the volume enclosed by the surface and not all three pairs of antipodal points can be found. When this happens, we simply treat the surface as a linear ellipsoid.

4. PATCH UNFOLDING

A class of traditional patch unfolding methods are based on discrete conformal mappings [Eck et al. 1995; Floater 1997; Lévy et al. 2002]. By fixing the texture coordinates of the boundary vertices, the texture coordinates of the interior vertices can be solved through a closed form system. These methods are fast and stable, and the solution is unique [Lévy et al. 2002]. However, conformal mappings do not preserve areas. Regions can be stretched or compressed, causing uneven sampling rates. Sander et al. [2001] have proposed a post-processing step in which the texture coordinates of the interior vertices are optimized to reduce a form of geometric stretch (which we will refer to as *Sander's stretch metric*), and their work inspired our own stretch optimization. We seek a definition of stretch that provides a balance between conformal mappings and area-preserving mappings.

4.1 The Green-Lagrange Tensor: a Balanced Stretch Metric

An *isometry* between two surfaces is a bijective mapping f that maintains distances between the two metric spaces, that is, $d(f(x), f(y)) = d(x, y)$ for all points x and y in the domain. An ideal surface parameterization P would be an isometry between the surface S and its images in the texture map I,

which means an everywhere even sampling is possible based on P. For most patches no isometric parameterization exists, except in the case of developable surfaces. Classical results from Riemannian geometry state that there exists a conformal ("angle-preserving") mapping between S and I. Some parameterization methods first compute a conformal parameterization for a patch, and then optimize the interior vertices based on some stretch metric [Sander et al. 2001; Lévy et al. 2002; Sheffer and de Sturler 2002]. Sander's metric (used in Sander et al. [2001]; Lévy et al. [2002]) helps balance the sampling given by the parameterization. Unfortunately, it does not always distinguish between isometries and anisotropic stretch. To illustrate this point and to introduce our new balanced stretch metric, we review Sander's metric and related background.

For a triangle $T = \{p_1, p_2, p_3\}$ in the surface $S \subset \mathbb{R}^3$, and its corresponding texture coordinates $U = \{u_1, u_2, u_3\}$ in $\mathbb{R}^2 = \langle s, t \rangle$, the parameterization $P : U \rightarrow T$ is the unique affine mapping that maps u_i to p_i $(1 \leq i \leq 3)$. To be more specific, let $A(v_1, v_2, v_3)$ be the area of the triangle formed by vertices v_1, v_2 and v_3, then

$$P(u) = \frac{A(u, u_2, u_3)p_1 + A(u_1, u, u_3)p_2 + A(u_1, u_2, u)p_3}{A(u_1, u_2, u_3)}. \tag{12}$$

Let P_s and P_t denote the partial derivatives of P. The metric tensor induced by P is:

$$G = \begin{pmatrix} P_s \cdot P_s & P_s \cdot P_t \\ P_s \cdot P_t & P_t \cdot P_t \end{pmatrix} = \begin{pmatrix} a & b \\ b & c \end{pmatrix}. \tag{13}$$

The eigenvalues of G are

$$\{\gamma_{max}, \gamma_{min}\} = \sqrt{\frac{(a + c) \pm \sqrt{(a - c)^2 + 4b^2}}{2}} \tag{14}$$

which represents the maximal and minimal stretch of a nonzero vector. Sander's metric is defined as the average stretch metric in all possible directions, that is,

$$L^2(T) = \sqrt{(\gamma_{max}^2 + \gamma_{min}^2)/2} = \sqrt{(a + c)/2}. \tag{15}$$

The metric has a lower bound of 1, and isometries achieve this lower bound.

Equation 12 assumes that the area of the triangle equals its textural image. When computing the global stretch, we assume the total area of the surface equals the total area of the textural image. This means that we need to add a global scale factor to each triangle. Let $A(t)$ and $A'(t)$ be the surface area and textural area of a triangle t, respectively. The global factor is

$$\rho = \sqrt{\frac{\sum_{t \in S} A(t)}{\sum_{t \in S} A'(t)}} \tag{16}$$

and we rewrite Equation 12 as:

$$P(u) = \frac{A(u, u_2, u_3)p_1 + A(u_1, u, u_3)p_2 + A(u_1, u_2, u)p_3}{\rho A(u_1, u_2, u_3)}. \tag{17}$$

Notice that individual triangles, in general, have scale factors different from ρ. Unfortunately under this scenario, there are anisotropic stretch for which Sander's stretch metric also gives a value of one. In particular, this metric cannot distinguish between isotropic and anisotropic stretch. For instance, all of the following tensors

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0.5 & 0 \\ 0 & 1.5 \end{pmatrix}, \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \tag{18}$$

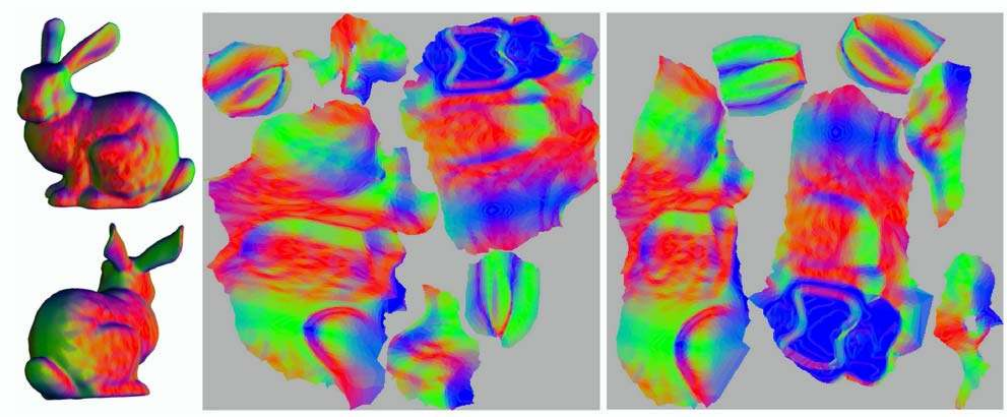

Fig. 11. This figure compares two surface parameterizations for the bunny obtained by vertex optimization based on Sander's stretch metric [Sander et al. 2001] (middle) and the Green-Lagrange tensor (right). Optimization based on Sander's metric causes high anisotropic stretch, especially the two largest patches. Compare the tail and the two rear legs (the square bumps on each side).

result in the same stretch measured in Sander's stretch metric, but the first one is clearly the most desirable. For this reason, we use the *Green-Lagrange tensor* to measure stretch and to guide patch optimization. Using the Green-Lagrange tensor as the stretch metric has been proposed before [Maillot et al. 1993]. However, it has not been used for patch optimization.

Using the above terminology, the *Green-Lagrange tensor* of G_t is defined as $\|G_t - I\|$, in which I is the identity matrix. The square of the *Frobenius norm* of this tensor is $T(G_t) = (\|G_t - I\|_{\mathbf{F}})^2 = (a-1)^2 + 2b^2 + (c-1)^2$. It is zero if and only if G_t is an isometry. We therefore define the stretch as

$$E_t^2 = 2T(G_t) = 2((a-1)^2 + 2b^2 + (c-1)^2) = [(a-c)^2 + 4b^2] + [(a+c-2)^2] = E_{conformal}^2 + E_{area}^2. \quad (19)$$

Notice that for the tensor to be conformal, we need $a = c$ and $b = 0$. When these conditions are met, the tensor becomes a scaling of magnitude $a = c$. It is an isometry if $a = c = 1$. This metric seeks a balance between the angle-preserving energy term $E_{conformal}^2$ and the area-preserving energy term E_{area}^2. A triangle's mapping is an isometry if and only if $E_t = 0$. This metric distinguishes between anisotropic stretch and isometries. In addition, it penalizes both undersampling and oversampling. However, the penalty is more severe for undersampling. This is desirable for texture mapping when a global isometry is not available. We note that Sorkine et al. [2002] devise a different stretch metric that also distinguishes anisotropic stretches from an isometry. We choose not to use their metric because it uses a max function, causing it to give equal stretch values to some cases that we feel should be distinguished.

The total balanced stretch of a patch S is therefore,

$$E^2(S) = \sum_{t \in S} \left\{ \left[(a_t - c_t)^2 + 4b_t^2 \right] + \left[(a_t + c_t - 2)^2 \right] \right\}. \quad (20)$$

The ideal value $E(S)$ for a patch S is zero, meaning all triangles in the patch are mapped isometrically.

Figure 11 compares the unfolding of the bunny surface using Sander's metric (middle) and the Green-Lagrange tensor (right). Notice that on the two largest patches, unfolding with Sander's metric produces anisotropic stretch (the tail and the two rear legs). The Green-Lagrange tensor performs well on all of these patches. Figure 10 shows the same comparison for the Venus. Again, optimization using Sander's metric causes anisotropic stretch. In Section 6.1, we will show the Green-Lagrange tensor also performs better in terms of image fidelity, despite sometimes having lower packing efficiencies.

4.2 Boundary Optimization with Scaffold Triangles

The process of *patch optimization* refers to moving vertices in the plane to minimize a given stretch metric. Most patch optimization methods handle the boundary vertices of a patch differently from the interior vertices. For an initial layout, boundary vertices are typically either mapped to the vertices of a convex polygon, or placed through conformal mappings. Sander et al. [2001] perform a nonlinear optimization on the interior vertices by moving one vertex at a time along some randomly chosen line to improve stretch. This process is very effective in reducing stretch during unfolding. Later, they extend the optimization framework to handle patch boundaries [Sander et al. 2002]. However, global foldovers may occur when a boundary vertex accidentally "walks" inside another triangle that is spatially faraway on the surface. To prevent this from happening, Sander et al. [2002] perform a global intersection test when performing optimization on a boundary vertex. However, this process is computationally expensive.

We introduce a new optimization method that allows the boundary vertices to move freely without the need to check for global foldovers. First, we compute an initial harmonic parameterization as described in Floater [1997]. Next, we construct a "virtual boundary" (a square) in the parameterization plane that encloses the patch. The 3D coordinates of the square are assigned to be mutually different and outside the convex hull of the patch in the 3D space. As we will see next, the exact coordinates of the virtual boundary are insignificant, provided that they do not coincide with each other or with the patch. Scaffold triangles are used to triangulate the region between the original patch boundary and the virtual boundary. Finally, we perform patch optimization [Sander et al. 2001] on the "enlarged" patch. There are two issues regarding scaffold triangles that need attention.

(1) How should we define stretch for scaffold triangles?
(2) How can we define and maintain their connectivity?

The first issue is handled as follows: the stretch of a scaffold triangle is defined as infinity if there is a foldover, otherwise it is defined as zero. This allows a boundary vertex to move within its immediate incident triangles to obtain better stretch without the need to check for global foldovers. Furthermore, the exact 3D coordinates of the virtual boundary are insignificant.

The second issue appears when the initial connectivity of scaffold triangles unnecessarily constrains the movements of boundary vertices. This is because scaffold triangles are designed to prevent global foldovers, that is, one patch vertex "walks" onto a patch triangle other than its immediate neighboring triangles, which unfortunately include the scaffold triangles. To remedy this overly conservative approach, we allow scaffold regions to be retriangulated at the end of each optimization step in which all the vertices have been moved. For any edge between two scaffold triangles, we perform an edge flip operation if it improves the triangles' aspect ratios.

Figure 12 (right) illustrates the effect of using scaffold triangles on an example patch on a cube (b: without scaffold triangles; d: with scaffold triangles). Notice that scaffold triangles allow the optimization to achieve a zero stretch in this case.

The shape of the virtual boundary and the connectivity of the scaffold triangles are insignificant since they merely serve as a placeholder to allow the boundary vertices of a patch to move freely without causing global foldovers. This is different from the work of Lee et al. [2002], in which virtual boundaries are constructed as parts of edge springs to obtain an initial parameterization. In their work, the shape and the connectivity of the virtual boundaries directly affect the stretch of the resulting parameterization. Indeed, several layers of virtual boundaries are often required to produce reasonable results using their method. In our work, only one layer is required.

Scaffold triangles also arise from hole-filling operations that occurred during genus reduction and feature identification. They are treated similarly as the scaffold triangles from the virtual boundary,

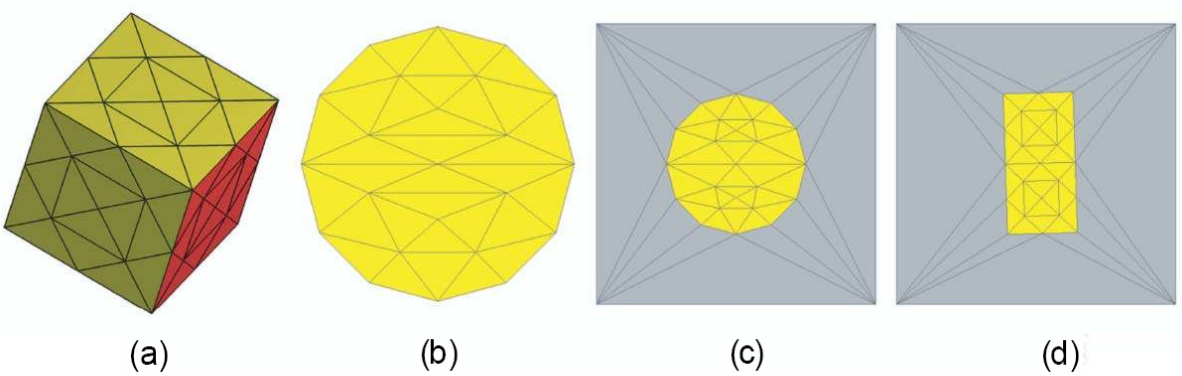

Fig. 12. This figure demonstrates the effect of scaffold triangles on patch unfolding. A patch that consists of two sides of a cube (a, colored in yellow) is unfolded using three methods: optimization without scaffold triangles (b), with scaffold triangles but without optimization (c), and with scaffold triangles and optimization (d). Scaffold triangles are colored in gray. Notice when both optimization and scaffold triangles are used, the patch is unfolded with a zero stretch.

that is, they do not contribute to the stretch metric unless there are foldovers, and their connectivity can be changed through retriangulation. Several of the patches in the texture maps shown in Figure 13 (right column, the dinosaur) and Figure 15 (bottom row, the dragon) have holes that make use of scaffold triangles.

5. PACKING

The final step of surface parameterization is *patch packing*, which refers to arranging unfolded patches inside a rectangular region (texture map) without any overlaps. The ratio between the total space occupied by the patches and the area of the rectangle is the *packing efficiency*. Higher packing efficiency indicates less wasted space in the final texture map. The problem of finding an optimal packing is a special instance of an NP-hard problem: *containment and minimum enclosure*. The problem has been studied extensively in the textile industry and the computational geometry community [Milenkovic 1998]. Several packing algorithms have been proposed as parts of some surface parameterization techniques [Sander et al. 2001; Lévy et al. 2002]. These methods are very effective when all the patches are nearly circular and have similar sizes. Since our patch creation technique tends to produce a small number of large and often-elongated patches, we developed a packing technique that takes into account the orientations of the patches and, in general, achieves better packing efficiencies. Later, we discovered that our method is very similar to the packing technique developed independently by Sander et al. [2003].

Our packing algorithm is based on the following two observations. First, our patch creation method tends to produce a small number of patches. Second, several of these patches are large and have elongated shapes. The first observation enables us to perform an optimal searching that would have been impractical for patch creation methods that produce hundreds of patches. The second observation indicates that the orientations of the large and elongated patches can help create gaps, into which smaller patches can be placed.

Our algorithm consists of two stages: initialization and placement. During the initialization stage, we create a "canvas", that is, an $N \times N$ grid structure at the textural resolution. Every cell in the canvas is marked as "unoccupied". Under the same resolution, we discretize the bounding box of every patch into a 2D grid and mark any cell intersecting the patch as "occupied". We obtain eight variations of grids for each patch by the combination of reflections with respect to the patch's vertical axis, the horizontal axis, and the diagonal.

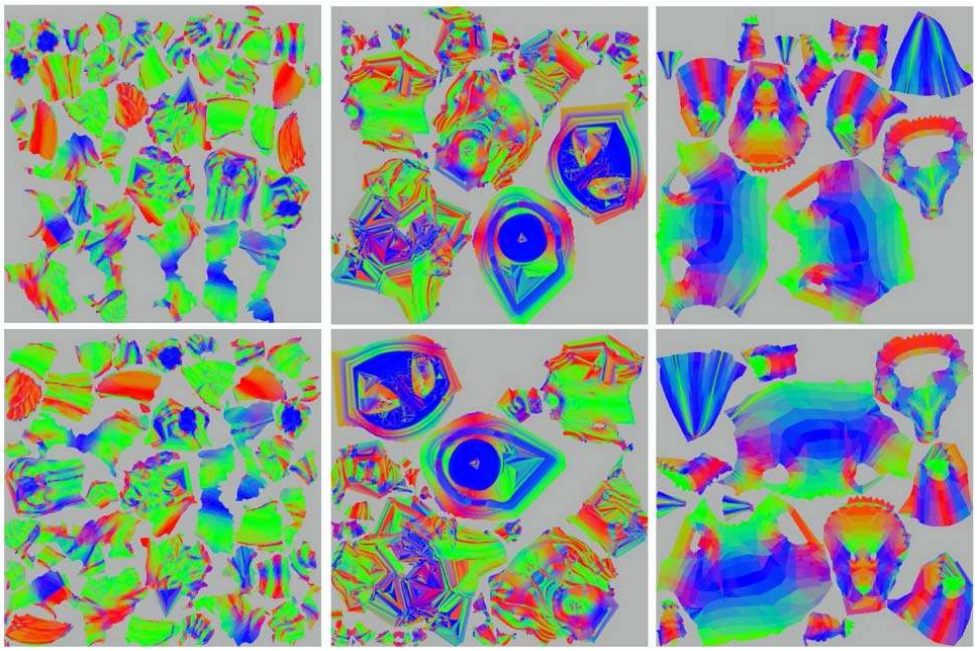

Fig. 13. This figure compares the packing results using the algorithm of Levy et al. [2002] (top row) and our algorithm (bottom row) for three models: the feline (left), the Buddha (middle), and the dinosaur (right). Notice the space under the "horizon" is used to pack smaller patches, and some patches are reflected diagonally to achieve a tighter packing.

During the placement stage, we insert the patches, one-by-one, into the canvas in the decreasing order of patch size (area). The first patch is placed at the lower left corner of the canvas. After a patch is inserted, we update the status of cells in the canvas that have been covered by the newly placed patch. Before inserting the next patch P_i, we examine its eight variations to find the one that minimizes the *wasted space* in the canvas. To be precise, let α, a $m \times n$ grid cells, be a variation for P_i. We wish to place the lower-left corner of α in the (a, b) grid cell in the canvas such that the following conditions are met:

(1) For any *occupied* grid cell (p, q) in α, the corresponding grid cell $(a + p, b + q)$ in the canvas is *unoccupied*.
(2) α minimizes $\max(a + m, b + n)$.

In other words, we wish to place the patch as close to the lower left corner of the canvas as possible. Once the best variation is chosen, we translate and scale the patch to reflect its position and orientation in the canvas.

After all patches have been inserted, usually only $M \times M$ grid cells in the canvas have become occupied. For all our test models, M is between one-third and one-half of the size of the canvas. We perform scaling to all patches with the same factor so that the $M \times M$ grid cells are mapped to $[0, 1] \times [0, 1]$.

Figure 13 shows the improvement of our packing algorithm (bottom row) over the algorithm by Lévy et al. [2002] (top row) with three test models: the feline (left), the Buddha (middle), and the dinosaur (right). Notice the space under the "horizon" is reused to pack small patches, and some patches are rotated/reflected to achieve a tighter packing.

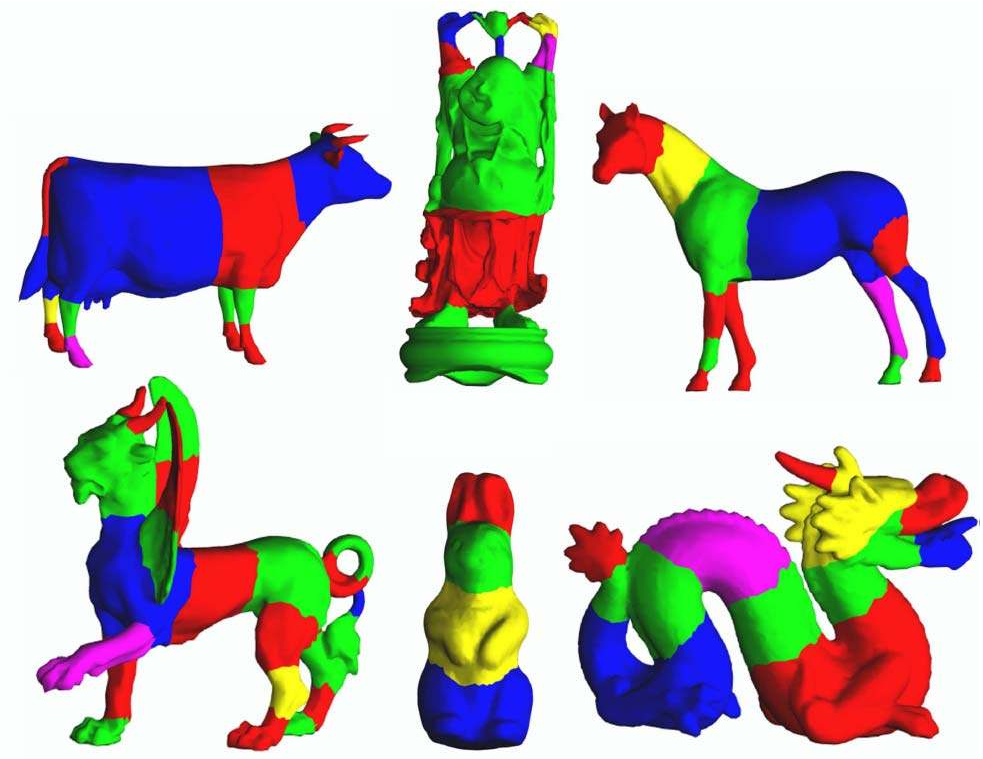

Fig. 14. This figure shows the result of our feature segmentation method on various test models. The cow, the horse and the rabbit are genus zero surfaces. The genus of the dragon, the Buddha, and the feline are one, six, and two, respectively.

6. RESULTS

We have applied our feature-based surface parameterization method to a number of test models. The results for the bunny and the dinosaur are shown in Figure 1 and 2, respectively. In Figure 14, we show the results of some other 3D models, including three surfaces with nonzero genus (the Buddha, the dragon, and the feline). Notice, in general, the feature regions are intuitive. For example, the horns and legs of animals are segmented from the bodies, and the Buddha's stand is identified as a single feature (a flat ellipsoid). (We wish to emphasize that no real animals were harmed during our research.) Figure 10 (right) and Figure 13 (lower middle) show the normal maps of the Venus and the Buddha, respectively. In a normal map, colors (R, G, B) are used to encode unit surface normals (x, y, z) [Sander et al. 2001]. Because of the many sharp creases on the Venus and the Buddha, patch creation methods based on surface curvature would have split the surfaces into many tiny patches, and such an example can be found in Lévy [2003]. Our method, however, was able to create large patches with little stretch.

Figure 15 shows textured models (left column) and the corresponding texture maps (right column) of the Buddha (top), the feline (middle), and the dragon (bottom). Table I gives the average stretch for the patches of the test models and the times for patch creation and unfolding using our method. The texture used for the Buddha is a wood texture from Perlin's noise [1985]. The textures used for the feline and the dragon were created by performing example-based texture synthesis directly on the surfaces [Turk 2001; Wei and Levoy 2001].

Fig. 15. This figure shows the parameterization of three models using our feature-based algorithm: textured models (left) and texture layouts (right, 512 × 512).

Table I. Average stretch (measured in Green-Lagrange) and timing results (minutes:seconds) for our patch creation and unfolding algorithm. Times are measured on a 2.4 GHz PC

model name	# polygons	# patches	stretch (Green-Lagrange)	patch creation time	patch unfolding time
Buddha	20,000	28	1.56	6:32	27:29
Buddha (large)	100,000	16	1.27	39:25	168:43
bunny	10,000	6	0.23	1:07	8:28
cow	10,524	29	0.28	2:15	4:01
dinosaur	10,636	14	0.25	1:37	5:53
dragon	20,000	24	0.83	3:00	16:23
dragon (large)	100,000	39	0.49	38:00	124:32
feline	10,000	41	0.22	2:31	2:32
feline (large)	100,000	46	0.29	32:57	109:27
horse	10,000	27	0.22	1:30	3:21
Venus	10,000	2	0.17	0:11	11:38
rabbit	10,000	8	0.24	0:53	4:50

6.1 Measuring Quality

Measuring the quality of a surface parameterization is an important yet complicated issue. It has several components.

(1) Stretch affects the sampling rate across the surface.
(2) Seams cause discontinuities across patch boundaries.
(3) Smoothness measures the amount of sharp changes in the sampling rates across interior edges of a patch.
(4) Packing efficiency determines the efficient use of the texture map.

When evaluating a surface parameterization method, it is not clear how these components should be combined measure the quality of the resulting map. On the other hand, for texture mapping applications, the quality of a surface parameterization should reflect "image fidelity", that is, the faithfulness of the images produced using the texture maps to the images for which the surface signals are directly computed. Next, we present an image-based metric, which draws inspiration from the work on image-driven mesh simplification [Lindstrom and Turk 2000].

Given a surface parameterization P, we first compute a continuous and smooth surface signal and store the result in a texture map based on P. Then, we render the surface from many viewpoints using the texture map, and compare the image differences with respect to the true surface signals. In practice, we choose 20 orthographic viewpoints that are the vertices of a surrounding dodecahedron. Let M_0 be the surface with the signal directly computed, and M_i be the textured surface with the texture size of $2^i \times 2^i$. The RMS "image" error between the images is calculated as:

$$RMS(M_i, M_0) = \sqrt{\sum_{n=1}^{20} D_i^n}. \tag{21}$$

Here, D_i^n is the squared sum of pixel-wise intensity difference between the n-th image of M_i and M_0. Equation 21 can be seen as the discretization of the following functional:

$$E(M, M_0) = \sqrt{\frac{\int_{\mathbf{p}\in S} D^2(M_0(\mathbf{p}), M(\mathbf{p}))V(\mathbf{p})d\mathbf{p}}{\int_{\mathbf{p}\in S} V(\mathbf{p})d\mathbf{p}}}. \tag{22}$$

Here M_0 and M refer to the original and the reconstructed surface signals (in colors), respectively. $D(M_0(\mathbf{p}), M(\mathbf{p}))$ is a perceptual metric between colors. For our application, we use

$$D((r_1, g_1, b_1), (r_2, g_2, b_2)) = \sqrt{((r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2)/3}. \quad (23)$$

$V(\mathbf{p})$ is the view-independent surface visibility as defined in Zhang and Turk [2002], which measures the visibility of $\mathbf{p}$ with respect to viewpoints on a surrounding sphere of an infinite radius. Therefore, $E(M, M_0)$ takes into account surface visibility in addition to color errors caused by stretch, seams, packing efficiencies, and smoothness of the parameterization. As demonstrated in Lindstrom and Turk [2000] and Zhang and Turk [2002], the error metric can be sampled with a small number of viewpoints that are evenly spaced in the view space.

One possible ideal surface signal can be obtained by first spreading a set of evenly spaced points on the surface and building a smooth function that uses these points as the bases. However, computing such a function over a surface with complex geometry is time-consuming. In contrast, a $3D$ checkerboard pattern is relatively easy to compute, and it has the nice property that the largest differential in frequencies in all directions is bounded. Although not perfect, it is nonetheless a good starting point. To make the signal continuous, we replace each "box" section with a "hat". The frequency in each main axial direction is the same. In practice, we use $1/16$ of the maximum side of the bounding box of the surface as the frequency.

Table II compares two unfolding methods, optimization with Sander's metric and the Green-Lagrange tensor for nine test models. Notice optimization with our metric produces lower stretch for all the test models. Furthermore, despite sometimes having lower packing efficiencies, optimization with our metric produces lower image errors for all the test cases. Figure 16 provides a visual comparison between the ideal signal (bottom-middle), the textured model using optimization with Sander's metric (bottom-left) and the Green-Lagrange tensor (bottom-right) for the Buddha model with the texture map of size 128×128. Notice the different level of blurring in the left image (front body and base) due to a highly uneven sampling rate. This phenomenon is less noticeable in the right image that uses our approach. Compare their corresponding texture maps: left row (Sander's metric) and right row (our technique).

7. CONCLUSION AND FUTURE WORK

In this article, we present an automatic surface parameterization method in which manifold surfaces are divided into feature regions for patch creation. By performing topological analysis of the average geodesic distance function, we are able to divide the surface into a small number of large patches that can be unfolded with little stretch. For patch unfolding, we use the Green-Lagrange tensor to guide the vertex optimization process. Also, we use scaffold triangles to allow the boundary vertices of a patch to be optimized. Although they were developed with texture mapping in mind, we think our surface segmentation and genus reduction methods might also be useful for other applications. Finally, we describe an image-based quality measure for surface parameterization techniques.

There are several areas for improvement. First, the nonlinear optimization stage of our patch unfolding algorithm is rather slow. Sander et al. [2002] propose a hierarchical optimization framework, which we would like to adapt to our system with scaffold triangles. Second, our feature identification technique sometimes creates more feature regions than necessary. For instance, the horse's legs are divided into two or three regions (Figure 14). While this does not seem to dramatically affect the quality of final texture maps, we are looking for alternative methods that can determine separating regions more robustly. For instance, we would like to see whether AGD could be used to decide the boundaries of separating regions. Since AGD is less noisy and is intrinsic to the surface, we expect that it contain more useful information.

Table II. This table compares two stretch metrics for guiding Sander-style vertex optimization. With the exception of the columns labeled "Image Error", the top row in each data cell are the results using Sander's metric, and the bottom rows are the results using the Green-Lagrange tensor (Section 4.1). For a comprehensive comparison, three measurements are provided: average stretch (the first two columns), packing efficiency (third column), and image-based error metric (Section 6.1, the last two columns). The numbers in the "Image Error" columns are the percentage of error difference of the image error caused by Sander's metric to the image error caused by our technique. For all nine test models, optimization with the Green-Lagrange tensor results in lower average stretch (either measured using Sander's metric or the Green-Lagrange tensor). Despite sometimes having lower packing efficiencies, our technique results in less image errors for all the test models

Comparison for patch unfolding with different optimization metrics (top row using Sander's metric [Sander et al. 2001], bottom row using Green-Lagrange tensor)					
	Stretch measured in Sander's metric	Stretch measured in Green-Lagrange	Packing Ratio	Image Error 128×128	Image Error 256×256
Buddha	1.27 1.18	26.80 1.56	0.67 0.68	8.28%	10.77%
bunny	1.13 1.02	3.92 0.23	0.60 0.65	14.46%	10.93%
cow	1.11 1.03	3.07 0.28	0.73 0.65	1.89%	1.62%
dinosaur	1.07 1.03	1.55 0.25	0.59 0.66	13.22%	5.16%
dragon	1.26 1.13	13.78 0.83	0.67 0.67	11.14%	12.84%
feline	1.10 1.02	1.73 0.22	0.66 0.64	7.25%	3.46%
horse	1.09 1.03	1.65 0.22	0.67 0.66	5.57%	0.93%
Venus	1.10 1.02	2.99 0.17	0.59 0.66	16.29%	15.26%
rabbit	1.12 1.03	3.00 0.24	0.68 0.65	8.73%	4.54%

Many additional topics in this area are interesting to us. First, our algorithm does not directly minimize the amount of seams caused by the segmentation. It would be desirable to have control over seams.

Second, although our image-based quality measure implicitly takes into account stretch, seams, smoothness, packing efficiency and visibility, it may be desirable to understand and control their impact on the quality of a surface parameterization. For instance, while the Green-Lagrange tensor is a good indicator of stretch, how it relates to the image-based quality measure deserves further investigation.

Third, we are still looking for other functions for surface segmentation. Our metric has been chiefly based on surface distance, which is intrinsic to the surface. Are there other functions that combine both intrinsic and extrinsic properties of the surfaces that might result in an even better segmentation? For instance, is there a surface function that helps find a separating cycle for the feline's wing exactly where human would place it?

Finally, surface parameterization is important for many applications, one of which is surface visualization. A complex surface often contains interiors and concavities that are difficult to see from the outside viewpoints. We would like to investigate the use of parameterization for surface exploration, giving the user the ability to navigate, orient, and focus.

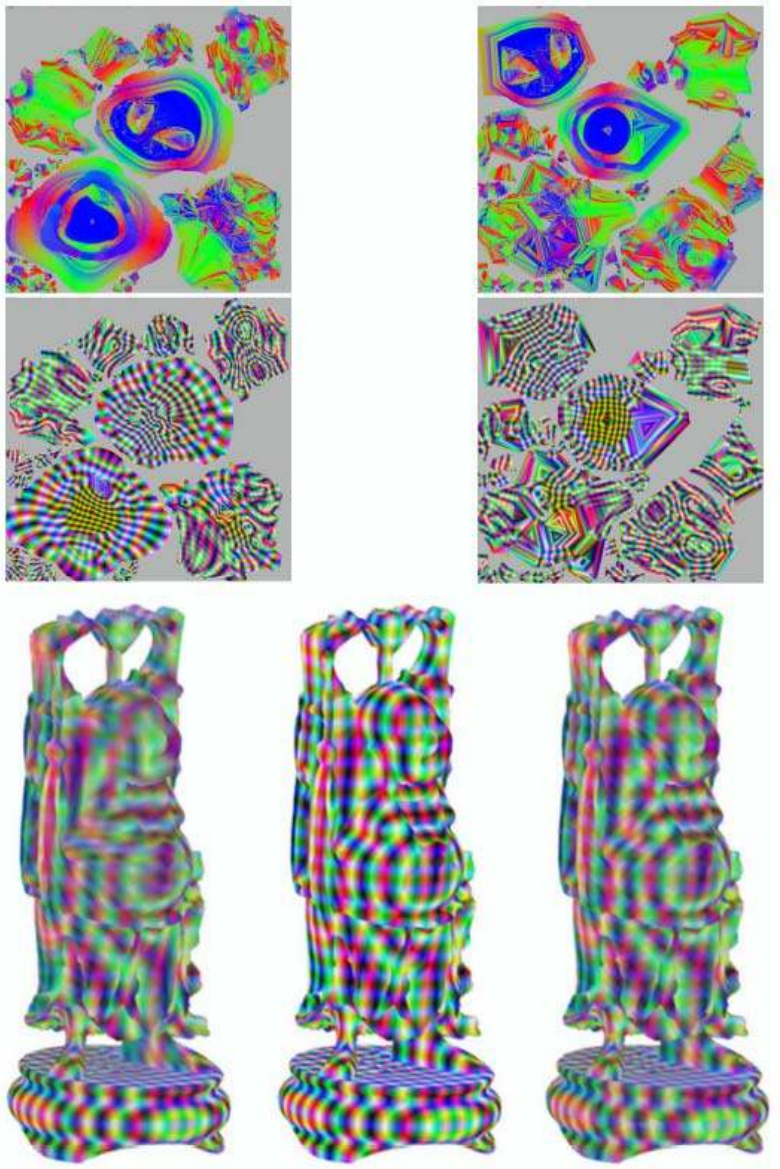

Fig. 16. Comparisons between patch unfolding with Sander's metric [Sander et al. 2001] (left column) and with Green-Lagrange stretch tensor (right column) for the Buddha using the 3D texture described in Section 6.1 (original signal is shown in the middle of the bottom row). In each column, from top-to-bottom, are the normal map, the map of the 3D texture, and the textured model. Notice in the left column (Sander's metric), the patches created from the base are assigned more space than those from the Buddha's torso. This causes a loss of signal in the textured model (face, body, and feet). On the other hand, optimization using the Green-Lagrange tensor (right column) produces a more even sampling rate and the reconstruction error is less noticeable.

ACKNOWLEDGMENTS

We would like to thank the following people and groups for the 3D models they provided: Zoë Wood and Peter Schröder, Mark Levoy and the Stanford Graphics Group, Andrzej Szymczak, and Cyberware. We also appreciate the discussions with Jarek Rossignac and Andrzej Szymczak. Finally, we wish to thank our anonymous reviewers for their valuable comments and suggestions.

REFERENCES

ALLIEZ, P., MEYER, M., AND DESBRUN, M. 2002. Interactive geometry remeshing. *ACM Trans. Graph. (SIGGRAPH 2002) 21*, 3 (July), 347–354.

AXEN, U. AND EDELSBRUNNER, H. 1998. Auditory morse analysis of triangulated manifolds. *Mathematical Visualization*, H. C. Hege and K. Polthier, Eds. Springer-Verlag, Heidelberg, Germany, 223–236.

BANCHOFF, T. F. 1970. Critical points and curvature for embedded polyhedral surfaces. *American Mathematical Monthly 77*, 475–485.

BENSON, D. AND DAVIS, J. 2002. Octree textures. *ACM Trans. Graph. (SIGGRAPH 2002) 21*, 3 (July), 785–790.

CARR, N. A. AND HART, J. C. 2002. Meshed atlases for real-time procedural solid texturing. *ACM Trans. Graph. 21*, 2, 106–131.

CIGNONI, P., MONTANI, C., ROCCHINI, C., AND SCOPIGNO, R. 1998. A general method for recovering attribute values on simplified meshes. *IEEE Visualization Proceeding*, 59–66.

DEBRY, D., GIBBS, J., PETTY, D. D., AND ROBINS, N. 2002. Painting and rendering textures on unparameterized models. *ACM Trans. Graph. (SIGGRAPH 2002) 21*, 3 (July), 763–768.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. *Proceeding of Eurographics*, 209–218.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1995)*, 173–182.

EDELSBRUNNER, H., HARER, J., AND ZOMORODIAN, A. 2003. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete Comput. Geom. 30*, 87–107.

ERICKSON, J. AND HAR-PELED, S. 2002. Optimally cutting a surface into a disk. *Symposium on Computational Geometry*, 244–253.

FLOATER, M. S. 1997. Parameterization and smooth approximation of surface triangulations. *Comput. Aid. Geomet. Design 14*, 3 (July), 231–250.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. OBB-Tree: A hierarchical structure for rapid interference detection. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1996)*, 171–180.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. *ACM Trans. Graph. (SIGGRAPH 2002) 21*, 3 (July), 355–361.

GUSKOV, I. AND WOOD, Z. 2001. Topological noise removal. *Graphics Interface*, 19–26.

HANRAHAN, P. AND HAEBERLI, P. E. 1990. Direct WYSIWYG painting and texturing on 3d shapes. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1990)*, 215–223.

HILAGA, M., SHINAGAWA, Y., KOHMURA, T., AND KUNII, T. L. 2001. Topology matching for fully automatic similarity estimation of 3d shapes. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 203–212.

HORMANN, K. AND GREINER, G. 1999. MIPS: An efficient global parameterization method. *Curve and Surface Design: Saint-Malo 1999*, Laurent, Sablonnière and Schumaker, Eds. Vanderbilt University Press. 153–162.

KACZYNSKI, T., MISCHAIKOW, K., AND MROZEK, M. 2004. *Computational Homology*. Applied Mathematical Sciences 157, Springer-Verlag.

KATZ, S. AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. (SIGGRAPH 2003) 22*, 3 (July), 954–961.

KHODAKOVSKY, A., LITKE, N., AND SCHRÖDER, P. 2003. Globally smooth parameterizations with low distortion. *ACM Trans. Graph. (SIGGRAPH 2003) 22*, 3 (July), 350–357.

KIMMEL, R. AND SETHIAN, J. A. 1998. Computing geodesic paths on manifolds. *Proceedings of National Academy of Sciences 95,* 15 (July), 8431–8435.

KINDLMANN, G. AND WEINSTEIN, D. 1999. Hue-balls and lit-tensors for direct volume rendering of diffusion tensor fields. *IEEE Visualization Proceeding*, 183–189.

LAZARUS, F., POCCHIOLA, M., VEGTER, G., AND VEROUST, A. 2001. Computing a canonical polygonal schema of an orientable triangulated surface. *17th ACM Symposium on Computational Geometry*, 80–89.

LEE, A., SWELDENS, W., SCHRÖDER, P., COSWAR, L., AND DOBKIN, D. 1998. MAPS: Multiresolution adaptive parameterization of surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1998)*, 95–104.

LEE, Y., KIM, H. S., AND LEE, S. 2002. Mesh parameterization with a virtual boundary. *Computers & Graphics (Special Issue of the 3rd Israel-Korea Binational Conference on Geometric Modeling and Computer Graphics) 26,* 5, 677–686.

LÉVY, B. 2003. Least squares conformal maps gallery. http://www.loria.fr/levy/Galleries/LSCM/index.html.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. (SIGGRAPH 2002) 21*, 3 (July), 362–371.

LINDSTROM, P. AND TURK, G. 2000. Image-driven simplification. *ACM Trans. Graph. 19*, 3, 204–241.

LOPES, H., ROSSIGNAC, J., SAFONOVA, A., SZYMCZAK, A., AND TAVARES, G. 2003. Edgebreaker: A simple compression algorithm for surfaces with handles. *Comput. Graph. Int. J. 27*, 4, 553–567.

MAILLOT, J., YAMIA, H., AND VERROUST, A. 1993. Interactive texture mapping. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1993)*, 27–34.

MILENKOVIC, V. J. 1998. Rotational polygon containment and minimum enclosure. *Proceedings of the 14th Annual Symposium on Computational Geometry,* ACM.

MILNOR, J. 1963. *Morse Theory*. Annals of Mathematical Studies. Princeton University Press, Princeton, NJ.

NI, X., GARLAND, M., AND HART, J. C. 2004. Fair morse functions for extracting the topological structure of a surface mesh. *ACM Trans. Graph. (SIGGRAPH 2004) 23*, 3 (Aug.), 613–622.

PERLIN, K. 1985. An image synthesizer. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1985)*, 287–296.

PIPONI, D. AND BORSHUKOV, G. 2000. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2000)*, 471–478.

REEB, G. 1946. Sur les points singuliers díune forme de pfaff completement integrable ou díune fonction numerique [on the singular points of a completely integrable pfaff form or of a numerical function]. *Comptes Randus Acad. Sciences Paris 222*, 847–849.

ROURKE, C. AND SANDERSON, B. 1972. *Introduction to Piecewise-Linear Topology*. Springer Verlag.

SANDER, P. V., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2002. Signal-specialized parameterization. *Proceedings of the 13th Eurographics Workshop on Rendering*, 87–100.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 409–416.

SANDER, P. V., WOOD, Z. J., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. *Proceedings of the 1st Symposium on Geometry Processing*.

SHEFFER, A. AND DE STURLER, E. 2001. Parameterization of faceted surfaces for meshing using angle based flattening. *Engineer. with Comput. 17*, 3, 326–337.

SHEFFER, A. AND DE STURLER, E. 2002. Smoothing an overlay grid to minimize linear distortion in texture mapping. *ACM Trans. Graph. 21*, 4, 874–890.

SHEFFER, A. AND HART, J. C. 2002. Seamster: Inconspicuous low-distortion texture seam layout. *IEEE Visualization Proceeding*, 291–298.

SORKINE, O., COHEN-OR, D., GOLDENTHAL, R., AND LISCHINSKI, D. 2002. Bounded-distortion piecewise mesh parameterization. *IEEE Visualization Proceeding*, 355–362.

TURK, G. 2001. Texture synthesis on surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 347–354.

WEI, L. Y. AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 355–360.

WOOD, Z., HOPPE, H., DESBRUN, M., AND SCHRÖDER, P. 2004. Removing excess topology from isosurfaces. *ACM Trans. Graph. 23*, 2, 190–208.

ZHANG, E. AND TURK, G. 2002. Visibility-guided simplification. *IEEE Visualization Proceeding*, 267–274.

Received June 2003; revised April 2004; accepted August 2004