International Journal of Shape Modeling
© World Scientific Publishing Company

Parameterization using Manifolds

Cindy M. Grimm

Dept. of Computer Science and Engineering,
Washington University in St. Louis,
One Brookings Dr.,
St. Louis, MO 63130, USA
cmg@wustl.edu
http://www.cs.wustl.edu/∼cmg

Received (1 December 2002)
Revised (15 May 2003)
Accepted (10 March 2004)

There are a variety of surface types (such as meshes and implicit surfaces) that lack a natural parameterization. We believe that manifolds are a natural method for representing parameterizations because of their ability to handle arbitrary topology and represent smooth surfaces. Manifolds provide a formal mechanism for creating local, overlapping parameterization and defining the functions that map between them. In this paper we present specific manifolds for several genus types (sphere, plane, n-holed tori, and cylinder) and an algorithm for establishing a bijective function between an input mesh and the manifold of the appropriate genus. This bijective function is used to define a smooth embedding of the manifold that approximates or interpolates the mesh. The smooth embedding is used to calculate analytical quantities, such as curvature and area, which can then be mapped back to the mesh.

Possible applications include texture mapping, surface fitting, arbitrary topology surface modeling, feature detection, and surface comparison.

Keywords: Splines; texture mapping; parameterization; n-holed tori

1991 Mathematics Subject Classification: 32C09

1. Introduction

There are many surface representations, such as meshes and implicit surfaces, that lack a “built-in” parameterization, such as the one provided by spline surfaces. The primary use of a surface parameterization in graphics is as a texture map. A parametric surface equation is also useful for calculating differential geometry entities such as geodesics and principal curvature. These metrics can then be used for applications such as feature extraction, shape classification, and comparisons of 3D objects.

Parameterization is essentially the problem of flattening a surface (or piece of a surface) to the plane without folding or creasing it. This creates a mapping from the surface to the plane. Current approaches with meshes have focused on finding “nice” mappings that distribute distortions in well-behaved ways [10,30,44,50,2]. Implicit surfaces have received less attention [37,51,17] but the idea is very similar. Surface fitting with a parameterized surface [32,25,11] also requires a mapping between the domain of the fitted surface and the data; most of these fitting algorithms contain a parameterization assignment and optimization stage before computing control points. All of these approaches first divide the surface into one (or more) planar regions, each of which is mapped to the plane.

The work presented here addresses a problem fundamental to all of the above parameterization optimizations — topology. If the surface does not have a planar (or toroidal) topology than the surface must be cut into one or more pieces that are planar. The most common approaches are to let the user define the cuts [38,25], to split along optimal lines [30,44], or to choose the seam lines based on a mesh simplification method such as progressive meshes [20,41,39]. These seams introduce discontinuities in the parameterization which the user must then hide or otherwise disguise.

A manifold representation addresses the topology problem by dividing the surface into *overlapping* regions, each of which is parameterized locally. Where the surface regions overlap we define functions which move between the overlapping regions. These functions can be C^0, C^k, C^∞, *etc.* This provides a structured mechanism for blending or moving between different parameterizations. C^∞ manifolds can also be built for any topology (including surfaces with boundaries) since they create multiple local parameterizations, not a global one.

Another property of a manifold representation is that it separates the topology of a surface from the geometry. The topology is defined by the way in which the parameterizations overlap. The geometry is defined by embedding the manifold in R^n. We demonstrate this property by using the same manifold representation, with different embeddings, to approximate several different surfaces. This has potential uses in surface comparisons because points on two surfaces can be related to each other through their parameterizations.

As an application example, consider segmenting a mesh M into areas of similar Gaussian curvature. This segmentation could be used to, for example, classify shapes in a database. We first parameterize the mesh using the appropriate manifold. Next, we define a *continuous* function on the manifold that returns the Gaussian curvature at each point. This function could be built by either embedding the manifold (Section 6) and analytically computing the curvature (Section 7), or by calculating the curvature at each vertex [34] and fitting a function (defined on the manifold) that interpolates or approximates those values. Given a point in the domain we can always find a local parameterization that contains both that point and a fairly large neighborhood around it. This means we never have to check to see if we have crossed a boundary or seam line and therefore need to use a “different” function.

In Section 3 we define manifolds in general, and discuss how to build a specific manifold representation that is amenable to implementation. In Section 4 we define specific manifolds for several topologies. These manifolds are designed to have several properties that support operations in the domain of the surface, for example, evaluating a metric over a local neighborhood.

The subject of Section 5 is how to establish a mapping from a mesh to a manifold of the appropriate topology. We first define what a valid mapping is (note that this mapping is at best C^0 since the mesh is C^0). Second, we provide an algorithm, based on existing parameterization techniques, for constructing the mapping.

Given a mapping from a mesh to the manifold we can next define an embedding of the manifold that approximates or interpolates the mesh (Section 6). The embedding is used to calculate (analytically) approximations to differential geometry entities, such as principal curvature and area, on the original mesh. We use these quantities in Section 7 to evenly distribute points on a surface, divide the surface into regions of uniform surface area, and trace lines along principal curvature directions.

2. Previous work

We cover related work in two areas: manifold representations and parameterization papers. There are also a wide variety of surface fitting techniques, some of which address parameterization indirectly. The interested reader is referred to the excellent survey paper by Lodha and Franke [32].

The majority of parameterization papers address the problem of texture mapping in particular, although a few have emerged from the finite element literature. The primary issue in parameterization is how to flatten a mesh (or portion of the mesh) into the plane while minimizing the resulting distortion. Techniques differ in how they measure distortion, the complexity (non-linear versus linear) and convergence guarantees of the optimization routine, whether or not they require a fixed boundary, and guarantees about folding or overlapping. Some methods also allow the user to influence the parameterization by adding point or gradient constraints. Some papers also focus on how to partition or cut the mesh so that the resulting pieces are minimally distorted.

One of the first texture mapping papers for meshes was by Maillot [33]. The mesh was first broken into an atlas based on regions with similar surface normals. The individual sections were flattened by optimizing for minimal edge length distortion (derived from an approximation of the first fundamental form), with an error term to reduce triangle flipping.

Lévy has pioneered work with user-constrained texture mapping [29,28], using optimization metrics based on conformal measures [30]. These techniques do not require a fixed boundary and do not suffer from flipping (although they may overlap). The original paper [29] used an iterative optimization scheme; this has been improved to a linear optimization in subsequent papers [28,30].

Floater [10] introduced several parameterization methods based on the idea of placing a vertex at the weighted centroid of its neighbors. These techniques are linear, and guaranteed not to fold if the boundary is fixed and convex. These techniques have been applied to surface reconstruction [12] and extended to arbitrary meshes [11] by the use of a base set of triangles to represent the topology of the mesh.

Sheffer and Sturler [42] introduced an angle-based flattening method, which minimizes the deviation in angles. It is guaranteed not to fold, although it may overlap itself. The optimization is based on a set of linear constraints on the angles which guarantee that the angles around a vertex sum to 360, the angles in a triangle sum to 180, and each disk around a vertex is a closed polygon. This work has been extended to reduce distortion in scale [45]. Sheffer also presents a technique to introduce seams in a mesh in order to reduce distortion. The original paper searched for minimal spanning trees [43] that joined vertices with high curvatures. Subsequent work uses minimal Steiner trees and a notion of visibility [44] to find ideal seams. The same seam-location approach can be used to constrain lines in texture maps [24] to particular paths on the mesh.

Another class of parameterization papers arise in the context of remeshing [11,8,40,26] and mesh compression [20]. The basic idea is to find a mapping from the triangles of a high-resolution mesh to the triangles of a coarse mesh approximation. The high-resolution mesh is partitioned into disk regions, each of which is mapped to a triangle in the coarse mesh. These approaches work on arbitrary topology meshes, but they do so by partitioning the mesh and solving the subsequent independent parameterization problems (with the exception of Praun [40]). Floater [11] optimizes the parameterizations using his existing method [10]. Eck et. al. introduce the Harmonic map optimization, which tries to maintain relative edge lengths. This technique requires a fixed boundary and has no guarantees about folding. Lee et. al. [26] iteratively minimize the angle deformation as vertices are introduced going from the coarse to fine level. In a similar manner, Praun et. al [40] iteratively reduce a stretch metric as vertices are added to the coarse triangulation. The stretch metric measures the ratio of the largest to the smallest stretch at a point. Both of these approaches use a local optimization which, in general, is not guaranteed to produce a global solution. Guskov [20] and Praun [41] both define techniques for finding paths in the high-res mesh that correspond to edges in the base mesh. These paths are optimized to produce good patch parameterizations.

Khodakovsy et. al. [23] produce a base set of patches which are individually parameterized, but unlike other approaches, they optimize their base set of patches along with the individual parameterizations. The parameterization of one patch also influences the neighboring patches, which reduces discontinuities across the boundaries.

Parameterization also arises in the context of mesh morphing and mesh alignment. For example, Alexa [1] embeds a spherical mesh using a relaxation approach. An initial (small) set of vertices are placed on the sphere, and the remaining vertices moved to the centroid of their neighbors. This method occasionally has problems if the initial pinned vertices are not sufficiently spaced to prevent collapse. Praun [41] used the same base polyhedron for multiple meshes, then found a mapping from the high-res meshes to the polyhedron. The initial alignment of the polyhedron was produced by hand, but the individual patch alignments were automatic.

Desbrun [6] presents a parameterization technique that is a direct extension of the conformal map error measures to a discrete surface. A variation of this method does not require a fixed boundary and has a unique global solution, although it has no guarantees about folding or overlapping.

Several approaches use geodesics to create parameterizations. The iso-parameter lines are constrained to lie along geodesics [2]. Bennis [3] used this approach to create better texture maps for B-splines. Pedersen [37] allowed a user to draw geodesics on implicit surfaces, and automatically generated a set of interior geodesics inside the patch to constrain the interior. Zigelman [50] uses geodesics to establish distances between points in a mesh and uses multi-dimensional scaling to embed those points in 2D. This method has no guarantees about folding.

Several papers approach the problem of parameterizing non-zero genus surfaces without partitioning them first. Haker et al. [21] present a conformal map for spherical topologies. They first embed the mesh in the complex plane by solving a second-order PDE (which can be formulated as a linear system for meshes), then use inverse stereo projection to map back to the sphere. Gu and Yau [19] extend conformal mapping to other topologies by using the appropriate periodic transforms; they also formulate an alternative to Haker's PDF which does not require an inverse stereo projection, although it is an iterative optimization algorithm. Gotsman, et. al. [14] present a generalization of the weighted centroid to the sphere. They define what a weighted centroid means, and specific numerical methods for solving for vertex locations on the sphere given a set of weights. In general, the difficulty in solving this problem is that there exists a degenerate solution (where all the vertices map to one point), unlike the planar case. Previous approaches [1,19,40] got around this problem by constraining or localizing some part of the optimization.

It should be noted that the parameterization issue is somewhat separate from the question of domain representation, which is the primary focus of this paper. Any of the previous papers that map a mesh to a sphere, for example, can be used to create the bijection between the spherical manifold and a genus one mesh. The planar parameterization methods are less directly applicable to surfaces of non-zero genus, although some of the distortion measurement concepts and user-interface constraints are applicable. One thing to note is that techniques that operate on arbitrary genus meshes by splitting the surface into disks [28,44,23] will, in general, have lower distortion within the local parameterization than the global approach. This is at the cost, however, of introducing discontinuities in the parameterization.

Several papers describe surface construction techniques using manifolds [15,16,35,36,31]. Grimm's approach [16] begins with a mesh and builds a manifold with one chart per mesh element. The approach in Navau and Garcia's first paper [36] builds a manifold for a planar mesh by mapping the boundary of the mesh to the unit square. Charts and embedding functions can then be built on the unit square. We adopt this approach for planar meshes. For arbitrary topology meshes Navau and Garcia extend this approach [35] by first subdividing the mesh to isolate extraordinary vertices. They embed sections of the mesh so that the overlap regions are rectangular and blend together in the middle in a C^k fashion. Subdividing the mesh to isolate the extraordinary vertices can result in a large number of patches; however, the patches themselves are simpler than the ones in Grimm [16]. Manifolds have also been used for smoothing subdivision surfaces [4] by creating charts for the region around a vertex and re-defining the embedding function on that chart.

The work in this paper is a continuation of previous work [15] in building simple manifolds for parameterization. We include two new manifolds; cylinders (a tube with two boundaries) and n-holed tori, the latter of which is described in detail in a forth coming paper [18]. We introduce an improved technique for creating a guaranteed initial bijection between the mesh and the manifold in the sphere and torus case. We also define an algorithm for creating an initial bijection for the n-holed torus and cylinder cases. We also address the problem of defining barycentric coordinates for non-planar topologies.

3. Manifold and atlas definitions

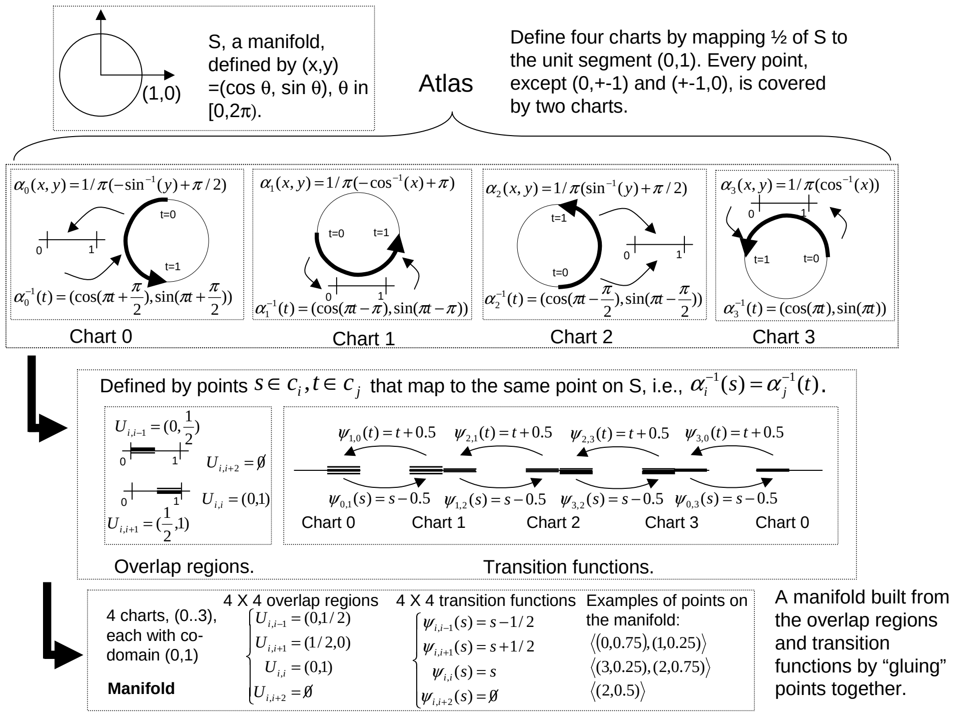
The original definition of a manifold can be found in the topology literature [47,27]. The basic idea is to analyze a complicated surface by defining maps[a] from the surface to R^2. Each of these maps takes an open disk of the surface down to an open disk in the plane, with no pinching or folding, and is called a *chart.* The collection of chart domains must completely cover the surface, *i.e.,* every point on the surface must be in the domain of one or more charts. The collection of charts is called the *atlas.* Any surface that can be described in this manner is considered to be a manifold.

As a real-world analogy, consider an atlas of the world. Each of the pages represent a mapping from the earth (which is a sphere, and hence a manifold) down to the plane. The pages overlap enough so that you can navigate from one page to an overlapping one (although in general the overlaps do not line up perfectly). For example, the page for France contains part of Spain, and vice-versa. The area that is shared between the two pages is called an *overlap* region. The function that maps from points in the France page to corresponding points in the Spain page is called a *transition* function. Figure 1 shows a 2D example of defining an atlas on the unit circle. This atlas defines a set of overlap regions and transition functions.

We now introduce definitions that formalize the concepts of atlas, chart, overlap regions, and transition functions:

- $\mathcal{S}$ is an existing manifold.

[a]Manifold theory is valid for a surface of dimension n embedded in a dimension $m > n$.



(1,0)
S, a manifold, defined by (x,y) =(cos θ, sin θ), θ in [0,2π).
Atlas
Define four charts by mapping ½ of S to the unit segment (0,1). Every point, except (0,+-1) and (+-1,0), is covered by two charts.
$\alpha_0(x,y)=1/\pi(-\sin^{-1}(y)+\pi/2)$
$\alpha_0^{-1}(t)=(\cos(\pi+\frac{\pi}{2}),\sin(\pi+\frac{\pi}{2}))$
Chart 0
$\alpha_1(x,y)=1/\pi(-\cos^{-1}(x)+\pi)$
$\alpha_1^{-1}(t)=(\cos(\pi t-\pi),\sin(\pi t-\pi))$
Chart 1
$\alpha_2(x,y)=1/\pi(\sin^{-1}(y)+\pi/2)$
$\alpha_2^{-1}(t)=(\cos(\pi t-\frac{\pi}{2}),\sin(\pi t-\frac{\pi}{2}))$
Chart 2
$\alpha_3(x,y)=1/\pi(\cos^{-1}(x))$
$\alpha_3^{-1}(t)=(\cos(\pi t),\sin(\pi t))$
Chart 3
t=0
t=1
0
1
Defined by points $s\in c_i, t\in c_j$ that map to the same point on S, i.e., $\alpha_i^{-1}(s)=\alpha_j^{-1}(t)$.
$U_{i,i-1}=(0,\frac{1}{2})$
$U_{i,i+2}=\emptyset$
$U_{i,i}=(0,1)$
$U_{i,i+1}=(\frac{1}{2},1)$
$\psi_{1,0}(t)=t+0.5$
$\psi_{2,1}(t)=t+0.5$
$\psi_{2,3}(t)=t+0.5$
$\psi_{3,0}(t)=t+0.5$
$\psi_{0,1}(s)=s-0.5$
$\psi_{1,2}(s)=s-0.5$
$\psi_{3,2}(s)=s-0.5$
$\psi_{0,3}(s)=s-0.5$
Chart 0
Chart 1
Chart 2
Chart 3
Chart 0
Overlap regions.
Transition functions.
4 charts, (0..3), each with co-domain (0,1)
Manifold
4 X 4 overlap regions
$U_{i,i-1}=(0,1/2)$
$U_{i,i+1}=(1/2,0)$
$U_{i,i}=(0,1)$
$U_{i,i+2}=\emptyset$
4 X 4 transition functions
$\psi_{i,i-1}(s)=s-1/2$
$\psi_{i,i+1}(s)=s+1/2$
$\psi_{i,i}(s)=s$
$\psi_{i,i+2}(s)=\emptyset$
Examples of points on the manifold:
$\langle(0,0.75),(1,0.25)\rangle$
$\langle(3,0.25),(2,0.75)\rangle$
$\langle(2,0.5)\rangle$
A manifold built from the overlap regions and transition functions by "gluing" points together.

Fig. 1. Top: An atlas, consisting of four charts, defined on the unit circle. Each chart is a map from 1/2 of the circle to a unit length segment. Middle: The atlas defines overlap regions and transition functions. ψ_{ij} is defined by first mapping from the chart i up to the unit circle using α_i, then back down to j using α_j^{-1}. All indices are to be taken modulo 4, $\sin^{-1}$ returns a value between $\pm\pi/2$ and $\cos^{-1}$ returns a value between 0 and π. Bottom: A topologically equivalent manifold.

- Let A be a finite set of maps from $\mathcal{S}$ to a disk in the plane. A is called an *atlas.* Each element $\alpha_c \in A$ is called a *chart.* The co-domain of each chart we label as $c \subset \mathrm{R}^2$. *Chart* can also refer to the co-domain.
- A set of subsets, $U_{ij} \subset c_i$, where α_{c_i} and α_{c_j} are charts in A and where $U_{ii} = c_i$. These are the overlap regions. U_{ij} may be empty. The atlas A defines U_{ij} in the following way. A point $p \in c_i$ is in U_{ij} iff there exists $q \in c_j$ such that $\alpha_{c_i}^{-1}(p) = \alpha_{c_j}^{-1}(q)$.
- A set of functions Ψ called *transition functions.* A transition function, $\psi_{ij} \in \Psi$, is a map $\psi_{ij} : U_{ij} \to U_{ji}$ where $U_{ij} \subset c_i$ and $U_{ji} \subset c_j$. Note that U_{ij} and U_{ji} may well be empty. The Ψ functions can be built from A by defining ψ_{ij} to be $\alpha_j \circ \alpha_i^{-1}$.

The atlases and charts found in the topology literature are primarily used to analyze existing surfaces, *i.e.,* given an existing surface S, build an atlas to show that S is a manifold. In Grimm [16] this definition was inverted in order to produce a manifold from a set of charts and transition functions, without using a pre-existing surface. Instead of starting with S and building an atlas, start by defining a set of chart co-domains, overlap regions, and transition functions. Now define a manifold from these objects by "gluing" together points that are the same. The resulting object is a manifold, provided certain conditions are met [16], but what is interesting is that it does not have an embedding in R^n, *i.e.,* there is no specific geometry associated with the manifold. The advantage of defining a manifold in this manner is that it is very amenable to implementation. The bottom of Figure 1 shows a manifold, built from just the overlap regions and transition functions, whose *topology* is a circle, but without any geometry. A point on this manifold is just a tuple of all of the points that were "glued" together using the transition functions. Note, though, that there exists a bijection between the manifold and the canonical surface used to build it, *i.e.,* every point on the circle corresponds to a unique point on the manifold and vice-versa.

- A point p on the manifold is expressed as a tuple of chart points, one tuple for each chart that contains the point p. A chart point is written as $[\alpha_c \in A, (x, y) \in c]$.

For this paper, all of the manifolds we build are of this form. We can always embed the manifold to produce a geometric shape (see Section 6 and Figure 7 for an outline of the process). In this way we can create several surfaces that have the same domain (underlying manifold), but different geometries. As a technical note, the continuity of a manifold built in this way is the continuity of the transition functions; all of the manifolds built in this paper are C^∞. The continuity of the embedded manifold will therefore be the continuity of the embedding function.

To define the overlap regions and transition functions we use the method outlined in Figure 1. We first choose a canonical manifold S of the appropriate topology, and then define an atlas on S. This atlas defines a set of charts co-domains, overlap regions, and transition functions. We then "throw out" S and build a manifold (by gluing) that is topologically equivalent, but is not tied to the geometry of S. A point on this new manifold is defined as just a tuple of chart points.

To ensure that the "glued together" object is a manifold, we essentially need to show that the transition functions are reflexive ($\psi_{ii}(p) = p$), symmetric ($\psi_{ij}(\psi_{ji}(p)) = p$), and transitive ($\psi_{ij} = \psi_{kj} \circ \psi ik$). Using an atlas defined on an existing manifold S trivially ensures that the transition functions meet this criteria.

4. Specific manifolds

In this section we define specific manifolds for use in parameterization. Our goal is to create manifolds that are easy to use, extensible, and simple. We define one manifold for each genus, rather than defining a unique manifold for each mesh. This pushes the complexity of the parameterization into the mapping between manifold and mesh, and not into the manifold itself.

The following are desirable properties for parameterizations:

- The co-domains of the charts are unit squares or disks (uniformality).
- The number of charts is kept small. Charts can always be added to an existing manifold by subdividing existing charts.
- The domains of the chart overlap in disk-shaped regions, and they overlap substantially (approximately one-third area). This maximizes the chance that a local surface operation is contained within a single chart.
- The ability to partition the co-domains of the chart so that every point on the manifold lies in exactly one chart partition. This is useful for operations such as tessellation.

We use a canonical surface to build the manifolds. For each genus there exists more than one possible atlas; we discuss our choice and describe other possible manifolds.

For each manifold we also discuss how to partition the manifold by defining a subset of the co-domain of each chart. Recall that, since the charts overlap, each point on the manifold lies in one or more charts. Some operations, such as tessellation, require that the manifold be partitioned into non-overlapping regions that exactly cover the manifold. We define these regions by defining, for each chart, a subset of the co-domain. More formally:

- Let $\{\nu_i \subset c_i\}_{c_i \in A}$ be a partition of the manifold. We call each subset ν_i the partition for chart i. Then for every point p on the manifold, $\alpha_{c_i}(p) \in \nu_i$ for exactly one chart i.

4.1. *Plane*

The plane manifold consists of a single chart defined on the uniform square. The partition of the plane manifold is just the single chart, *i.e.,* $\nu_0 = c_0$.

4.2. *Sphere*

The minimal number of charts needed to cover the sphere is two, one on each pole, that overlap in a ring around the equator. To create an atlas with n charts, choose n evenly distributed points on the sphere and project the disk centered on that point to the plane. There are two standard methods for parameterizing a disk of the sphere; stereographic projection and the latitude-longitude approach. Sterographic projection preserves circles centered around the projection point. The latitude-longitude parameterization is well-behaved around the equator but produces increasingly more distortion around the poles.

We decided to use six copies of the latitude-longitude equation for our atlas, one at each pole (see Figure 2). Each chart covers almost a half of the sphere. Six charts is the best compromise between maximizing overlap, minimizing distortion in the parameterizations, keeping the number of charts small, and maintaining symmetry.

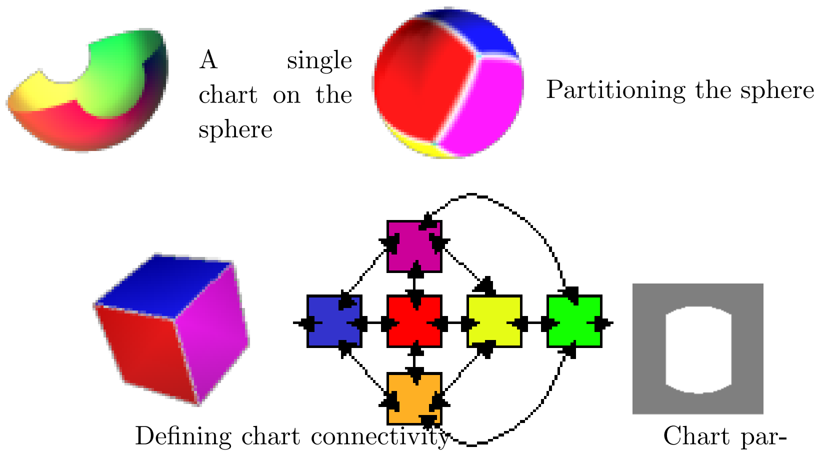


Fig. 2. Building charts for the sphere. We use a cube to define the adjacency relationships between the charts.

Also, we can use great arcs to partition the manifold into six equal regions (see Figure 2). Because we use the latitude-longitude equations the great arcs map to easily defined arcs in the chart co-domains.

$$\theta = u\pi, \quad \phi = v\frac{3\pi}{4} - \frac{3\pi}{8} \tag{1}$$
$$\alpha_0^{-1}(u,v) = \big(\cos(\theta)\cos(\phi), \sin(\theta)\cos(\phi), \sin(\phi)\big) \tag{2}$$
$$\alpha_1^{-1}(u,v) = \big(\cos(\theta+\pi)\cos(\phi), \sin(\theta+\pi)\cos(\phi), \sin(\phi)\big) \tag{3}$$
$$\alpha_2^{-1}(u,v) = \big(\sin(\theta)\cos(\phi), \sin(\phi), \cos(\theta)\cos(\phi)\big) \tag{4}$$
$$\alpha_3^{-1}(u,v) = \big(\sin(\theta+\pi)\cos(\phi), \sin(\phi), \cos(\theta+\pi)\cos(\phi)\big) \tag{5}$$
$$\alpha_4^{-1}(u,v) = \big(\sin(\phi), \cos(\theta)\cos(\phi), \sin(\theta)\cos(\phi)\big) \tag{6}$$
$$\alpha_5^{-1}(u,v) = \big(\sin(\phi), \cos(\theta+\pi)\cos(\phi), \sin(\theta+\pi)\cos(\phi)\big) \tag{7}$$

The inverse of these functions can be calculated using the appropriate arctan functions. We give the functions in pseudo C code (**atan2** returns the arc tangent in the range $\pm\pi$ for the input (y, x)).

$$\alpha_0(x,y,z) = \left(\frac{\text{atan2}(y,x)}{\pi}, (\arcsin(z) + \frac{3\pi}{8})\frac{4}{3\pi}\right) \tag{8}$$
$$\alpha_1(x,y,z) = \left(\frac{1+\text{atan2}(y,x)}{\pi}, (\arcsin(z) + \frac{3\pi}{8})\frac{4}{3\pi}\right) \tag{9}$$

The transition functions are built by taking $\psi_{ij} = \alpha_j \circ \alpha_i^{-1}$; for example:

$$\phi_{20}(u,v) = \Bigg(\frac{\text{atan2}(\sin(\frac{6\pi v - 3\pi}{8}), \sin(u\pi)\cos(\frac{6\pi v-3\pi}{8}))}{\pi}, \tag{10}$$

$$(\arcsin(\cos(u\pi)\cos(\frac{6\pi v - 3\pi}{8})) + \frac{3\pi}{8})\frac{4}{3\pi}\Bigg) \tag{11}$$

The overlap regions $U_{0,1}, U_{1,0}$, $U_{2,3}, U_{3,2}$, $U_{4,5}, U_{5,4}$, and their corresponding transition functions, are empty.

To produce a partition of the sphere manifold we use six great circles, which are the white lines on the sphere in Figure 2. This produces a slightly "bowed" rectangle with straight sides in each chart. Because we are using the latitude-longitude equations, each partition boundary is a straight line in some chart. We chose the chart equations so that the straight lines are always the vertical ones, *i.e.,* the chart partitions are identical for all charts. The start and stop points are determined by where the arcs intersect. The equations for the vertical lines are:

$$(0.25, t \in (\pm\frac{4sin^{-1}(\sqrt{1/3})}{3\pi} + 1/2)) \tag{12}$$

$$(0.75, t \in (\pm\frac{4sin^{-1}(\sqrt{1/3})}{3\pi} + 1/2)) \tag{13}$$

To find the upper and lower boundaries of the partition region we map the straight line from the overlapping chart into the current one using the appropriate transition function.

4.3. *Torus*

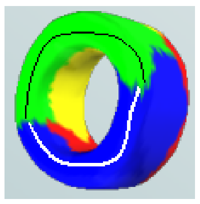
We use the standard embedding of the torus that takes the square $\theta \in [0, 2\pi), \phi \in [0, 2\pi)$ to the torus, but we only take 2/3 of the mapping for each chart:

$$T(\theta,\phi) = \big((\frac{3}{2} + \cos(\theta))\cos(\phi), (\frac{3}{2} + \sin(\theta))\cos(\phi), \sin(\theta)\big) \tag{14}$$

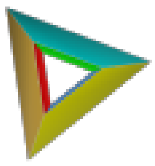
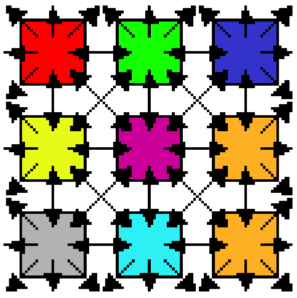
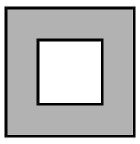
The minimal number of charts needed is one, but this provides for no overlap. The next obvious choice is a 2 by 2 grid, or four charts. We decided against this because any pair of charts overlaps at both ends, and we prefer simple disk overlaps. We settled on nine charts, each of which overlaps $2/3 \times 2/3$ of the torus function's domain. Numbering with chart zero in the lower left corner and two in the lower right corner we have:

$$\alpha_c^{-1}(s,t) = T\big((\frac{(c \bmod 3)}{3} + 2s/3)2\pi, (\frac{(c/3)}{3} + 2t/3)2\pi\big) \tag{15}$$

The inverse of this function is straightforward but requires some care with the bounds. We give the definition in pseudo C code:



A single chart Torus partition Four disks

Defining chart connectivity Chart parti-
tion

Fig. 3. Building charts for the torus.

$$r = ||(x, y)|| - 1 \tag{16}$$

$$\theta = \text{atan2}(z, r) \tag{17}$$

$$\phi = \text{atan2}(y, x) \tag{18}$$

$$u = \begin{cases} \frac{\theta}{2\pi} & \frac{\theta}{2\pi} < 0 \\ \frac{\theta}{2\pi} + 1 \; otherwise \end{cases} \tag{19}$$

$$v = \begin{cases} \frac{\phi}{2\pi} & \frac{\phi}{2\pi} < 0 \\ \frac{\phi}{2\pi} + 1 \; otherwise \end{cases} \tag{20}$$

$$s = \begin{cases} (u + 1 - \frac{(c \bmod 3)}{3}) & (c \bmod 3) = 2, u < \frac{1}{2} \\ (u - \frac{(c \bmod 3)}{3}) & otherwise \end{cases} \tag{21}$$

$$t = \begin{cases} (v + 1 - \frac{(c/3)}{3}) & \frac{c}{3} = 2, u < .5 \\ (u - \frac{(c/3)}{3}) & otherwise \end{cases} \tag{22}$$

The torus transition functions are all translations by $(\pm\frac{1}{4}, \pm\frac{1}{4})$.

To produce a partition of the torus we take the interior $[\frac{1}{4}, \frac{3}{4}) \times [\frac{1}{4}, \frac{3}{4})$ of each chart. This tiles the domain of the torus.

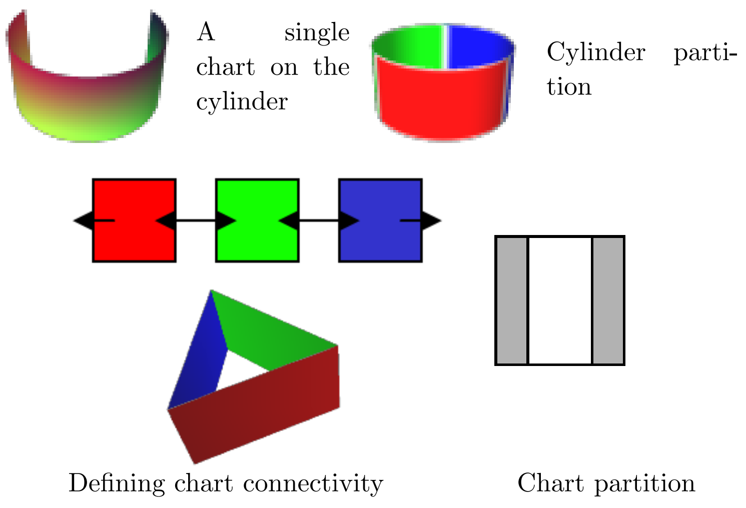


Fig. 4. Building charts for the cylinder.

4.4. *Cylinder*

The cylinder we define has no end-caps but is open at both ends (if it had end caps it would be topologically a sphere). The cylinder is built from three charts, which wrap in the u direction and have a boundary in the v direction (see Figure 4). The canonical surface is a unit radius, unit height cylinder centered at the origin $(C(\theta,h) = (cos\theta, h-0.5, sin\theta))$.

$$\alpha_c^{-1}(s,t) = C\big((\frac{(c \bmod 3)}{3} + s)2\pi, t-0.5\big) \tag{23}$$

$$\theta = \text{atan2}(z,x) \tag{24}$$

$$u = \begin{cases} \frac{\theta}{2\pi} & \frac{\theta}{2\pi} < 0 \\ \frac{\theta}{2\pi} + 1 & otherwise \end{cases} \tag{25}$$

$$s = \begin{cases} (u + 1 - \frac{(c \bmod 3)}{3}) & (c \bmod 3) = 2, u < .5 \\ (u - \frac{(c \bmod 3)}{3}) & otherwise \end{cases} \tag{26}$$

$$t = y + 0.5 \tag{27}$$

The cylinder transition functions are all translations by $(\pm\frac{1}{4}, 0)$.
The cylinder is partitioned by taking the middle stripe of each chart $[\frac{1}{4}, \frac{2}{4}) \times [0,1]$.

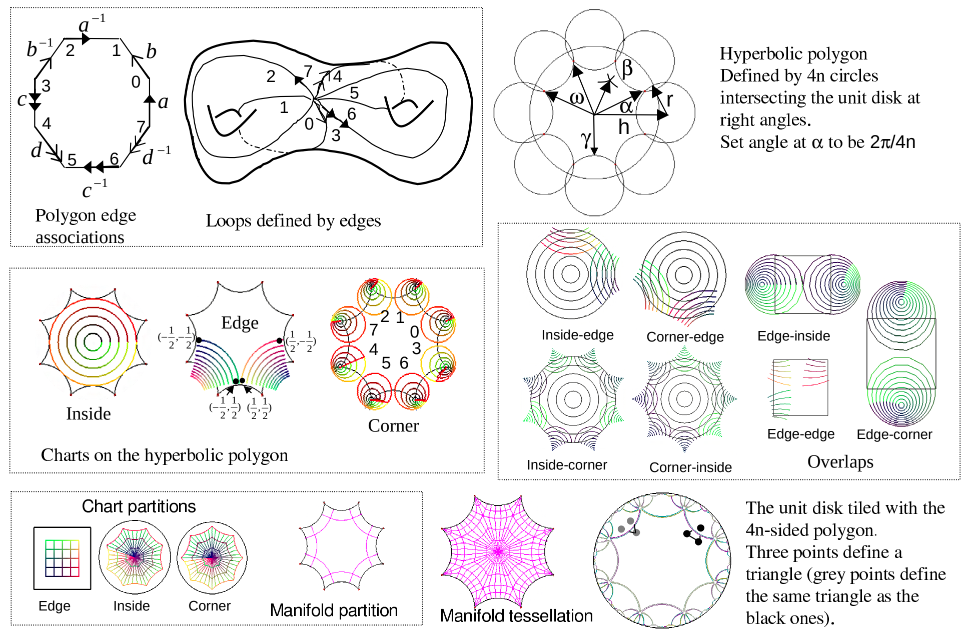

Polygon edge associations
Loops defined by edges
Hyperbolic polygon
Defined by 4n circles intersecting the unit disk at right angles.
Set angle at α to be 2π/4n
Inside
Edge
Corner
Charts on the hyperbolic polygon
Inside-edge
Corner-edge
Edge-inside
Inside-corner
Corner-inside
Edge-edge
Edge-corner
Overlaps
Chart partitions
Edge
Inside
Corner
Manifold partition
Manifold tessellation
The unit disk tiled with the 4n-sided polygon.
Three points define a triangle (grey points define the same triangle as the black ones).

Fig. 5. Building charts for the n-holed torus. The manifold is a hyperbolic polygon with $2\pi/(4n)$ corners (shown upper right) with edges associated as shown upper left. Top middle shows a two-holed torus with the boundaries of the polygon marked. Left middle shows the charts on the polygon; although the edge and corner chart are shown as several piece-wise mappings, when the polygon edges are associated those pieces will be joined together. Middle right: The overlap regions are one or more disks; the edge-edge overlaps may be one or three disks.

4.5. *N-holed tori*

We provide a very concise version of the n-holed torus here; a more detailed description can be found in an upcoming paper by Grimm and Hugues [18]. The canonical manifold is the hyperbolic polygon with associated edges. From topology, we know that an n-holed tori can be built from a $4n$-sided polygon by associating pairs of edges (see Figure 5). Consecutive sets of four edges define a handle, with one loop passing around the handle, the other one through the hole.

The hyperbolic polygon can be covered with a single chart, as in the case for the torus, but this provides for no overlap. We instead cover each associated edge-pair with one chart, place one chart in the inside of the polygon, and another that covers the corners of the polygon, for a total of $2n + 2$ charts. (Note that the corners of the polygon become a single point on the n-holed tori.) The edge chart co-domains are unit squares centered at the origin, while the corner and inside charts are unit disks, centered at the origin.

The overlap regions, unfortunately, are not single disks but instead are the union of from one to $4n$ disks, as can be seen in Figure 5. This means that a single transition function is built from several different maps, one for each disk. (Note that this does not affect the continuity of the transition functions, so long as each of the individual maps are continuous.) Fortunately, all of the maps are Linear Fractional Transforms (LFT):

$$\psi_{ij}(z) = Mz \tag{28}$$

where M is a 2×2 matrix representing the LFT and z is a complex number. LFTs operate on the complex plane, taking circles and lines to circles and lines, have a well-defined inverse, and the composition of two LFTs is a LFT. Each chart map is one or more LFT that rotates and translates a subset of the polygon to a subset of the unit square or disk. The two types of LFT we use are as follows (the first rotates and scales, the second translates):

$$S(p) = \begin{bmatrix} p & 0 \\ 0 & 1 \end{bmatrix} \qquad T(p) = \begin{bmatrix} 1 & p \\ p & 1 \end{bmatrix} \tag{29}$$

We define the transition functions as before, by first defining the chart maps to the manifold (left middle of Figure 5) then use composition to define the transition functions. Some care must be taken, however, to combine the correct combination of LFTs. The location in the co-domain of the source chart determines which map to the polygon to use. We then map the point to the polygon and determine where in the domain of the destination chart the point lies. Once we have built the transition functions the location in the source chart uniquely determines which transition function to use.

First some definitions (refer to the top right of Figure 5):

$$\omega = \frac{2\pi}{4n} \tag{30}$$

$$h = \sqrt{\frac{\cos(\omega) + 1}{\cos(\omega) + 1 - 2\sin^2(\omega/2)}} \tag{31}$$

$$r = \sqrt{h^2 - 1} \tag{32}$$

$$\Omega = h\cos(\omega/2) - \sqrt{(h\cos(\omega/2))^2 - 1} \tag{33}$$

$$\gamma = h - r \tag{34}$$

The inside chart is just a scale:

$$\alpha_i(c + \mathbf{i}d) = S(1.8\gamma)(c + \mathbf{i}d) \tag{35}$$

The edge chart is split into two maps, one for each side of the polygon edge (j is the index of the edge chart, numbered counter-clockwise in the polygon):

$$p_l = \cos((4j + j \bmod 2)\omega) + \mathbf{i}\sin((4j + j \bmod 2)\omega) \tag{36}$$

$$p_r = \cos((4j + 2 + j \bmod 2)\omega) + \mathbf{i}\sin((4j + 2 + j \bmod 2)\omega) \tag{37}$$

$$p_f = \cos\pi + \mathbf{i}\sin\pi \tag{38}$$

$$\alpha_{e_j}(c + d\mathbf{i}) = \begin{cases} S(p_l)T(h-r)S(7/8)(c + d\mathbf{i}) & c < 0.5 \\ S(p_r)T(h-r)S(7/8)S(p_f)(c + d\mathbf{i}) & c \geq 0.5 \end{cases} \tag{39}$$

The vertex chart is split into $4n$ maps, one for each corner of the polygon. The ordering of the wedges in the chart is *not* the same as the ordering of the polygon corners. Let w_j be wedge j in the chart, where wedge zero is centered on the negative real axis and the wedges are ordered in the clockwise direction. Let C_j be the polygon corner associated with wedge j. The polygon corners are indexed counter-clockwise, with the first corner at $\omega/2$. R_j is how much we need to rotate wedge j in order to correctly align it with the polygon corner.

$$R_{j+1} = R_j + 1 \tag{40}$$

$$C_{j+1} = \begin{cases} (C_j - 3) \bmod 4n & (C_j \bmod 4) = 1, 2 \\ (C_j + 1) \bmod 4n & (C_j \bmod 4) = 0, 3 \end{cases} \tag{41}$$

$$p_c = \cos((C_j + 1/2)\omega) + \mathbf{i}\sin((C_j + 1/2)\omega) \tag{42}$$

$$p_r = \cos(-R_j\omega) + \mathbf{i}\sin(-R_j\omega) \tag{43}$$

$$\alpha_{w_j}(c + \mathbf{i}d) = S(p_c)T(\Omega)S(p_r)S(\delta)(c + \mathbf{i}d) \tag{44}$$

where δ is a scale factor related to β in Figure 5; we chose δ so that the radius 0.45 circle passes through β. The point β is found by solving for s in the following equation:

$$\alpha_{e_0}(-s, s) = \alpha_{e_1}(-s, -s) \tag{45}$$

which ensures that the corners of all of the edge chart partitions (which lie at $(\pm s, \pm s)$) map to the same point β in the polygon.

To partition the manifold we create a rectangular partition in each edge chart $(\pm s, \pm s)$, and a wheel region in the corner and inside charts. The wheel boundaries are found by mapping the edges of the edge chart boundaries into the inside and vertex charts. This partitions the manifold into $2n+2$ regions, as shown in Figure 5.

5. Mapping

We first define what a valid mapping is, then define the algorithm for creating a mapping for each genus.

5.1. *Definition*

We want to establish a bijective mapping $\mathcal{M}$ between every point on a mesh $\{V, E, F\}$ and the corresponding manifold. The points of a triangular mesh[b] can be described uniquely by:

- A set of vertices V, $v \in \Re^3$.
- A set of edges E with $e(v_i, v_j) = (1-t)v_i + tv_j, \quad 0 < t < 1$.
- A set of faces F with $f(v_0, v_1, v_2) = \sum_i \beta_i v_i, \quad 0 \leq \beta_i \leq 1, \quad \sum_i \beta_i = 1$.

We first establish a map between the vertices of the mesh and the manifold. For each vertex v_i we chose a unique manifold point $p = \{(c_1, (s_1, t_1)), \ldots, (c_n, (s_n, t_n))\}$. We then extend this mapping to the edges and faces using barycentric coordinates and triangles in the manifold. In essence, each triangle in the mesh is mapped to a triangle in the manifold.

5.2. *Triangles and Barycentric Coordinates in the Manifold*

A triangle on the manifold is defined by three points p_i in the manifold. A point in the interior of the triangle is defined by the barycentric coordinates $0 \leq \beta_i \leq 1, \sum \beta_i = 1$. Exactly how this interior point is calculated is different for each topology type. For the plane, cylinder, and torus manifolds we choose an appropriate overlapping chart and use the standard linear sum:

$$\beta_i p_i = \alpha_c^{-1}\big(\sum_i \beta_i \alpha_c(p_i)\big) \tag{46}$$

Because the transition functions for these manifolds are affine transformations, this yields the same manifold point regardless of which chart we use.

For the sphere and the n-holed tori manifolds we define the triangles in the corresponding canonical surfaces (sphere and hyperbolic polygon). There are many possible definitions of triangles on the sphere; a recent paper by Praun [40] contains a summary. We use the Gnomonic triangle mapping, which is invertible. Each of the triangle points p_i are mapped to the sphere using any of the overlapping charts ($P_i = \alpha_c^{-1}(s,t)$ for any tuple $[c, (s,t)] \in p_i$). The three points P_i define a planar triangle in R^3 whose vertices lie on the sphere. The point $P = \sum_i \beta P_i$ in the planar triangle is projected to the sphere by normalizing by the length of $||P - (0,0,0)||$ (*i.e.,* the ray through the point P to the sphere).

For the n-holed tori we construct triangles in the Poincaré disk, then map the triangle to the equivalent Klein-Beltrami model [27] (where line segments are straight lines), compute the linear sum, then map the point back to the Poincaré disk. To determine the points in the Poincaré disk we find a chart that contains all three points, then choose *one* of the LFTs that map from the chart back to the Poincaré

[b]If the mesh is not triangular then we triangulate first.

disk and apply it to all three points. This may place the points outside of the center hyperbolic polygon, but because the hyperbolic polygon tiles the unit disk (see bottom right of Figure 5) the triangle simply "wraps" around the edge.

5.3. *Valid Mapping*

To ensure that the mapping does not fold we must guarantee the following:

- The mapping must preserve the ordering of the star of each vertex.
- The mapped edges must not intersect each other.
- For every face there must exist at least one chart into which all three of the vertices map.

For the plane, cylinder, and torus we use barycentric coordinates and a planar triangle to extend the map to the edges and faces. Let $\mathcal{M}_c(v) = (c, (s, t))$ be the map that takes a vertex to the chart c that contains all of the faces' vertices. We define the edge and face maps as convex combinations:

$$\mathcal{M}_c(e) = (1 - t)\mathcal{M}_c(v_i) + t\mathcal{M}_c(v_j) \tag{47}$$

$$\mathcal{M}_c(f) = \sum_i \beta_i \mathcal{M}_c(v_i) \tag{48}$$

The transition functions of the plane, cylinder and torus preserve linear combinations. This, with the properties listed above, guarantees that the mesh to manifold map is injective. The torus map will be surjective as well, since leaving a portion of the manifold uncovered would require folding the map at some point.

For the cylinder and plane case we must add an additional constraint that the boundary vertices map to boundary points on the manifold.

Inverting these functions is simply a matter of finding a triangle in some chart that contains the point and using the barycentric coordinates to map back to the mesh.

For the sphere we use Gnomonic triangles instead of planar ones. We first take the mesh and embed it in a unit sphere by placing the vertices at their designated locations:

$$v' = \alpha_c^{-1}(\mathcal{M}_c(v)) \tag{49}$$

where α_c^{-1} is the original chart map from the chart to the unit sphere. If we have chosen vertex locations that produce a bijection there are several things that hold true about this embedding. First, none of the mesh faces intersect. Second, the face normals all point out (or in). Third, if we take a ray from the origin to the boundary of the sphere it will intersect the embedded mesh in exactly one point.

The following is equivalent to Equation 48, except that we use ray casting through the point in the embedded triangle to find the equivalent point on the sphere:

$$\mathcal{M}_c(e) = \frac{(1-t)\alpha_c^{-1}(v'_i) + t\alpha_c^{-1}(v'_j)}{||(1-t)\alpha_c^{-1}(v'_i) + t\alpha_c^{-1}(v'_j)||} \tag{50}$$

$$\mathcal{M}_c(f) = \frac{\sum_i \beta_i \alpha_c^{-1}(v'_i)}{|| \sum_i \beta_i \alpha_c^{-1}(v'_i)||} \tag{51}$$

To invert the map we find the embedded triangle that the ray to the point on the sphere intersects. There is a simpler method than the one listed above for verifying that the choice of vertex locations produces a bijection — simply make sure that the face normals for all of the embedded triangles point out (or in).

For the n-holed tori we map each vertex to the hyperbolic polygon as in Equation 49. We then tile the Poincaré disk with the hyperbolic polygon [9] which also replicates the points v' in the disk. This means that there are multiple ways to form a triangle from the mapped points v'_i; choose the smallest one. Essentially, this means that faces that cross the boundary of the hyperbolic polygon will use points that lie outside of the center hyperbolic polygon (see bottom right of Figure 5). The entire Poincaré disk is then mapped to the Klein-Beltrami model, where edges are straight lines. At this point, we can apply the standard planar barycentric equations (Eq. 48).

To verify that the faces are oriented correctly in the disk, we simply check that all of the normals are $(0, 0, 1)$ (or $(0, 0, -1)$).

5.4. *Building manifold mappings*

To map the vertices from a mesh to a manifold we first partition the mesh into one or more disks. Each of these disks is mapped to a corresponding chart partition of the manifold, which has a known boundary. We use Floater's algorithm [10] with his shape-preserving weights for this mapping because it guarantees no folding with a convex boundary, and produces minimal-distortion embeddings. This produces a valid, bijective mapping, which we can then adjust if desired.

There are three basic mesh algorithms we use; a shortest path, a grow region, and a projection.

Grow disk: Takes a disk in the mesh and expands it by adding faces adjacent to the boundary edges. To ensure that the result is still a disk we only add a face if the boundary edges and vertices the face contains are continuous in the current boundary list. When the disk is the appropriate size we can optionally run an additional routine that takes out any chinks or "fins" in the boundary (see Figure 6).

Shortest path: Given two vertices, find the shortest path (in number of edges) between the two vertices. We may mark a subset of the vertices as not accessible.

Project: Given a disk in the mesh, and locations (s, t) in the chart for the boundary vertices, find locations for the interior disk vertices. We use Floater's algorithm [10] which requires solving an $n \times n$ matrix equation, where n is the number

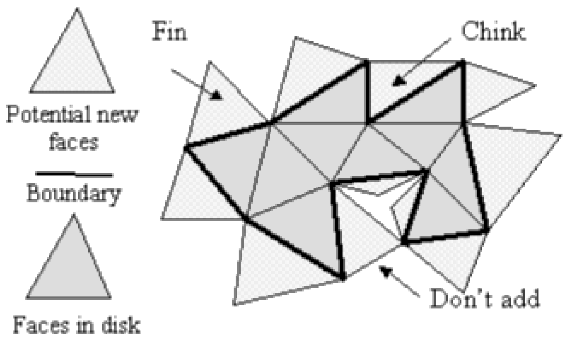


Fig. 6. A disk with its current boundary. All faces that share a boundary edge are potential faces to add. We do not add faces whose boundary is not contiguous along the disk boundary. When the disk is finished we can optionally remove "fins" and "chinks" by removing or adding the marked face.

of vertices. There is one linear equation for each boundary vertex ($\mathcal{M}_c(v) = (s,t)$) and one equation for each interior vertex ($\mathcal{M}_c(v) = \sum w_i \mathcal{M}_c(v^*)$) that places each vertex at the weighted centroid of its neighbors. We use Floater's technique to calculate shape-preserving weights; this calculation only needs to be performed once for each mesh since the weights do not change.

Once we have a valid mapping we can "slide" it around by moving the locations of the vertices, provided the new location of a vertex is within the convex hull of the polygon defined by the star of the vertex. Grimm [15] provides one method for minimizing distortion which involves re-projecting and re-scaling portions of the parameterization in an attempt to more evenly distribute the faces across the charts.

We introduce two alternative methods for reducing distortion. The first is to run a relaxation method where each vertex is moved slightly in a direction that minimizes some measure of distortion. Some care must be taken to ensure that the new locations still produce a bijection. We run this relaxation method only a small number of times (typically five) then check and re-project any offending vertices using the re-project method described next. We currently move each vertex towards the weighted centroid of its neighbors.

Re-project Our second method involves "cutting out" a convex portion of the mesh and re-projecting it using the **project** method described above. We define a convex region in some chart (typically an ellipsoid) and fix all vertices with edges that cross the boundary of the convex region. These become the boundary vertices in the **project** routine. The remaining vertices inside the convex region are then re-projected.

5.4.1. *Plane*

We use the **Grow disk** routine to find the boundary of the disk. We then evenly space the boundary vertices along the edges of the uniform chart, making sure there is a vertex that maps to each corner. We then use the **Project** routine to find vertex locations for the interior vertices.

5.4.2. *Cylinder*

We first find the top and bottom boundaries of the mesh. These are mapped to the $t = 1$ and $t = 0$ chart boundaries. We next split along a path from the top boundary to the bottom boundary. The resulting edges are mapped to $s = 0$ and $s = 1$. This produces a unique position for each vertex at $C(2\pi s, 2\pi t)$. We then use the **re-project** routine along the seam.

5.4.3. *Torus*

Find two paths that start and end at the same vertex, one that goes around the hole and one that goes through it. The first path must be a loop that cannot be contracted into a single vertex [27]. To produce this loop we grow four disks, each of which is seeded with a face that is as far as possible from the other three faces. We then look for two disks that meet in two or more disjoint boundary segments (Figure 3 shows an example torus with four example disks). We require that one half of the loop go through one disk and the second half go through the other disk, meeting at a mutually shared vertex from each of the disjoint boundary segments. The second path is found by choosing a vertex on the first path and finding a path that starts and stops at that vertex and does not cross the first path anywhere else. This is a variation of Dey's algorithm [7] which is proven to find the cut lines of the torus.

Now cut the mesh along the two paths and map the result to the unit square. This produces a unique position $T(2\pi s, 2\pi t)$ on the torus for each vertex. To reduce the effect of the artificial boundary we then run the **re-project** method around the seam boundaries, starting at the join point.

5.4.4. *Sphere*

First we **grow** a disk that has approximately 1/6 of the total surface area of the mesh. We **project** this disk into the partition of the first chart (which corresponds to 1/6 of the area of the sphere). We then project the remainder of the mesh (which is topologically a disk) into a rectangle using the same spacing along the boundary as the first projection, but in reverse order. We next place a quadrilateral inside of this rectangle, joining each corner of the quadrilateral to its corresponding corner of the rectangle. This creates five quadrilaterals, which correspond to our remaining five charts. We move the vertices of the primary quadrilateral until the sum of the mesh surface area of the triangles covered by each of the five quads is approximately 1/6 of the total surface area.

We run the **re-project** method along the seams (four edges and four corners).

5.4.5. *N-holed tori*

It is possible to find the loops that pass through and around the handles automatically [46] using co-homology groups, but in general this is very expensive for large meshes. We resort to user-intervention to define the loops; once these are located, the mesh is split open along the loops and mapped to the hyperbolic polygon, again using Floater's [10] algorithm. Note that the hyperbolic polygon is not convex, and therefore the solution may fold; this is an area for future work.

Again, we run the **re-project** method to reduce the effect of seams.

6. Embedding

To embed the manifold we first define for each chart c both a planar embedding function $E_c : c \to \mathrm{R}^3$ and a blend function $B_c : c \to \mathrm{R}$. We require that the blend function and its derivatives be zero by the boundary of the chart, and that the blend functions form a partition of unity. The planar embedding function can be of any form; we define two types, one based on hierarchical splines [13] and one based on radial basis functions.

The surface embedding is:

$$E(p) = \sum_{c \in A} B_c(\alpha_c(p)) E_c(\alpha_c(p)) \tag{52}$$

where $\alpha_c(p)$ extracts the R^2 point for chart c, if there is one. If p does not contain a point for chart c then B_c is defined to be zero. (Remember that a point p in the manifold is just a list of chart-points.)

To build C^k blend functions that form a partition of unity [16] we first build a proto-blend function in each chart using spline basis functions. We then "upgrade" these proto-blend functions to blend functions on the manifold by defining them to be zero for every manifold point not in the chart. This is why the proto-blend function, and its derivatives, must be zero by the boundary of the chart. To create a partition of unity we divide by the sum of all of the blend functions at that point; to ensure that this sum is non-zero we define the proto-blend functions so that their support covers the entire chart.

For the blend functions on the unit square co-domains we use the tensor product of two spline basis functions, each of which has a support of $(-1/2, 1/2)$ or $(0, 1)$. For the unit disk co-domains we use the distance from the chart center and a single spline basis function whose support is the diameter of the chart. This produces a radially symmetric blend function.

If the blend and embedding functions are C^k continuous then the resulting surface is C^k. To produce a visually pleasing surface it is best if the individual chart embeddings agree where they overlap, *i.e.*, $E_i(\alpha_i(p)) = E_j(\alpha_j(p))$; this ensures that the blending above is purely a formality.

We have experimented with two types of embedding functions, splines (approximating) and radial basis functions (interpolating). Approximating approaches are preferable when the data points are known to have noise in them; interpolating approaches tend to over-fit this type of data, producing ripples in the surface. Figure 7 shows curvature plots on the bunny mesh for the two different approaches. Notice that the spline embedding, while less accurate, also has less variation in curvature.

We use the same basic approach for both embedding types. First, each chart embedding function is chosen to minimize the error between the embedding function and the vertices within that chart:

$$\min(\sum_i ||E_c(\mathcal{M}_c(v_i)) - v_i||) \tag{53}$$

This lets us define each chart embedding function independently of the others. The total error will be at most the sum of all of the errors in the individual charts, scaled by their blend functions.

Fitting each chart is a standard parametric data interpolation problem [32], except that the density of the points within the chart will, in general, be very uneven. For this reason we proposed the use of hierarchical splines [13] in a previous paper [15]. In this paper we present an interpolating fitting technique using radial basis functions, which are very well adapted to this type of scattered data interpolation.

6.1. *Radial basis function embedding*

Radial basis functions have both advantages and disadvantages for surface modeling. They can be C^k to C^∞, depending upon the choice of kernel, they interpolate the input points, and are amenable to scattered data interpolation. Disadvantages are that they require n kernel functions to interpolate n data points and the surface may vary wildly between data points, especially in the presence of noise. There are techniques for dealing with noise and reducing the number of kernel functions needed. This produces an approximation, not an interpolation [5].

We use the following formulation:

$$RBF(s,t) = a_0 + a_1 s + a_2 t + \sum_i w_i d^2 log(d) \tag{54}$$

where the $a_i \in \mathrm{R}^3$ and $w_i \in \mathrm{R}^3$ are the variables, and $d = ||c_i - (s,t)||/2$. This kernel minimizes thin plate energy. The c_i are the parameter-space coordinates of the data points (in our case, the location of the vertex in the chart). The w_i are

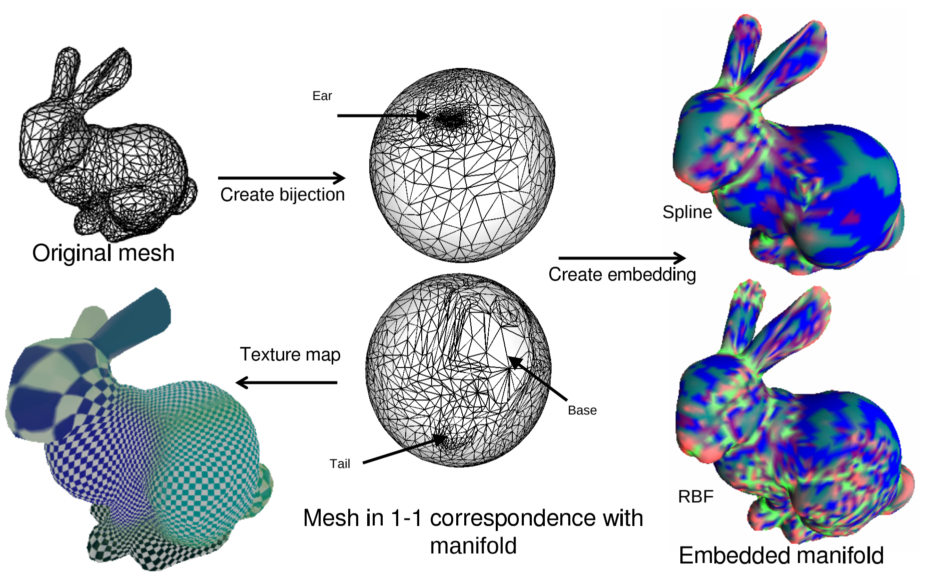


Fig. 7. The bunny mesh is first embedded in the sphere, creating a bijection between the mesh and the manifold. On the right are two possible embeddings, one defined using splines (approximating), the other using RBFs (interpolating). The embeddings are colored by Gaussian curvature g: blue is zero curvature $(0,0,1), |g| < 0.05$), red is positive curvature scaled by two $(2g,0,0)$, and green is negative curvature $(0,-2g,0)$.

weights assigned to each kernel function. The a_i are the coefficients of a one-degree polynomial. Solving for the w_i and a_i involves solving a matrix equation [5].

Figure 8 shows several example embeddings using radial basis functions.

7. Algorithms on the embedded manifold

The embedding of the manifold can be used to analytically calculate quantitative measures such as curvature and area. The resulting data can be mapped back to the original mesh, using the bijective mapping. These quantities are useful for the operations described below, as well as applications such as surface segmentation and comparison. For example, segmentation requires curvature (usually Gaussian) defined at each point on the surface. These points are then grouped into regions of similar curvature. Many image processing techniques exist for segmenting 2D images; we can easily extend these to a manifold by running the algorithm in the individual charts. Because the charts overlap, we can always find a local chart that contains a given point, and a neighborhood around that point, in which to perform the desired operation. We also have a well-disciplined method for propagating results from one chart to all overlapping charts.

We first show a sample derivative calculation to provide an example of the complexity of the calculation. Second, we describe a standard algorithm for distributing

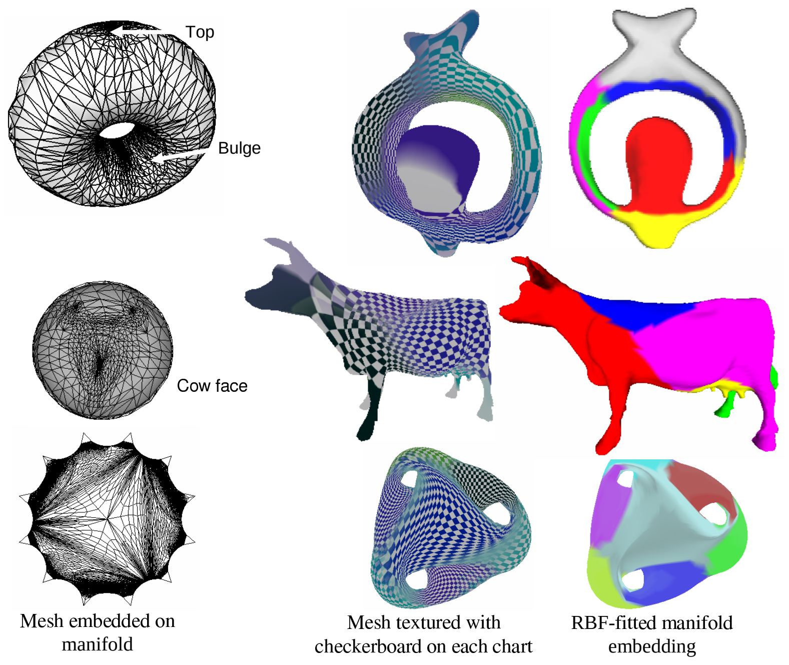

Top
Bulge
Cow face
Mesh embedded on manifold
Mesh textured with checkerboard on each chart
RBF-fitted manifold embedding

Fig. 8. Example radial basis function embeddings of different topologies. The original mesh is shown texture mapped with a checkerboard for each chart; the transparency of the checkerboard is set to the blend function B_c.

points on the surface. Third, we define an algorithm for finding regions of approximately equal surface area. Fourth, we extend the algorithm to tracing lines that follow the principle curvatures. All of these algorithms were originally developed for planar, parameterized surfaces. We demonstrate the extension of these algorithms to arbitrary topology surfaces via manifolds.

7.1. *Derivatives*

The calculation of derivatives, curvature, surface normals, and area is a straightforward application of partial derivatives. We outline the calculation of a derivative to give a feel for the amount of calculation involved. For readers unfamiliar with manifolds we point out that derivatives are taken with respect to a particular parameterization; in our case we may have several parameterizations at any given point. Surface normal, curvature, and area will return the same result regardless of the parameterization used, but derivatives will not.

The derivative of the embedding equation at a point (s, t) in chart c is the derivative of the sum of blended embedding chart functions. This reduces to a sum over only the overlapping charts:

$$\frac{\partial E(\alpha_c^{-1}(s,t))}{\partial s} = \partial(\frac{\sum_{c_i} E_{c_i}(\psi_{c,c_i}(s,t))B_{c_i}(\psi_{c,c_i}(s,t))}{\sum_{c_i} B_{c_i}(\psi_{c,c_i}(s,t))})/\partial s \tag{55}$$

where $\psi_{c,c_i}(s,t)$ is the transition function that takes the point $(s,t) \in c$ to the point $(s',t') \in c_i$. We will use the notation $\psi^0(s,t)$ to refer to the s component and $\psi^1(s,t)$ for the t component. Notice that the sum is only taken over the charts that overlap the point $(s,t) \in c$.

7.2. *Point distribution*

The goal of this algorithm is to place a set of points P on the surface such that any surface point is within distance d of some point $p_i \in P$ and every point in P is at least distance d from all other points. There are several ways of measuring distance. We currently measure distance along the line in parameter space connecting the two points.

$$\int_0^1 \mathcal{A}(t(u_0,v_0) + (1-t)(u_1,v_1))dt$$

Where $\mathcal{A}$ is the differential area at a point. If the two points do not lie in a shared chart then we choose an intermediate chart and split the problem into three segments. Note that, for our manifolds, the maximum number of intermediate charts we need is one. Which intermediate chart to use can be determined by the overlap topology; if there is more than one choice we choose the one with the shortest parameter length.

One approach to distributing points is to place them on the surface and "push" them away from each other, adding and deleting points as needed [48][49]. The basic algorithm looks like this:

```
While points too close or point missing neighbor
    For each point
        Find neighbors encircling point
        Move toward center of neighbors
        If closest less than current minimum distance
            Save pair as closest pair
        If gap in neighbors
            Add point in gap
    If minimum distance less than some threshold
        Delete one of closest pair
```

To keep this algorithm stable, creation and deletion should only be allowed on every n^{th} iteration, where n is around ten. Otherwise there is a tendency to delete the recently created points and vice-versa.

The distance calculation is easier if every pair of points has a mutual chart; in general this will be the case because a single chart is valid for approximately $\frac{1}{3}$ to $\frac{1}{2}$ of the surface.

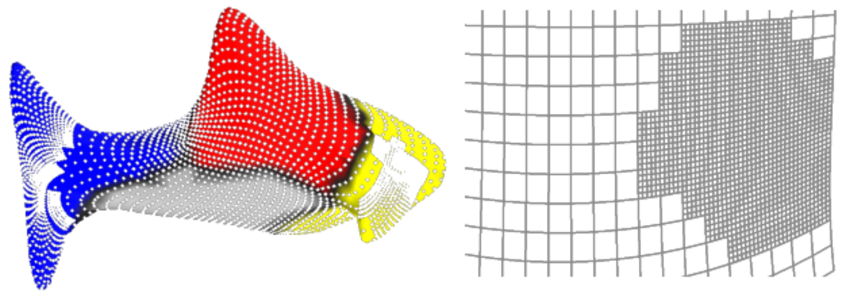


Fig. 9. Left: The surface with the center of the grid dots shown. Right: The grid cells.

"Gap" determination is performed in parameter space. We take all of the points that lie within some distance d' of the input point and map them to a single chart. We choose d' so that the distance on the surface between the point $E(< c, (s,t) >)$ and $E(< c, (s \pm d', t \pm d') >)$ is approximately twice the desired spacing d. Each neighbor resides at some angle (in parameter space) with respect to the input point. A gap exists if the spacing between two adjacent angles is greater than some threshold, usually taken as $\frac{3}{4}\pi$.

7.3. *Regions of similar area*

The goal is to divide the surface area into roughly circular areas of a desired radius. Even surface divisions are useful for re-meshing and texture mapping. This is accomplished using a greedy, iterative algorithm. The first step is to pick a point and grow a circular region of radius $d/2$ around it, where d is the desired spacing. Starting at any point that touches this region, we grow another circular region, placing the new point in the center of the new region. We continue until every part of the surface is in some region.

Starting with the centers of the current regions we grow circles of increasing radius (up to $d/2$), until the circles bump into each other. Any unclaimed region that is big enough receives a new point. Any region which is too small is deleted. The points are then moved to the center of their regions and the process is repeated.

To implement this algorithm we chop up the surface (by marking out squares in the domain) into regions each of which has less than $\pi d^2/10$ area. We do this using an n-ary partition of the center of each chart (see Figure 9). For each square we keep track of the closest neighbors in each of the eight directions (there may be multiple neighbors if the adjacent square was subdivided further). Growing disks is accomplished using a flood-fill algorithm.

Figure 10 shows several surfaces with points distributed on them.

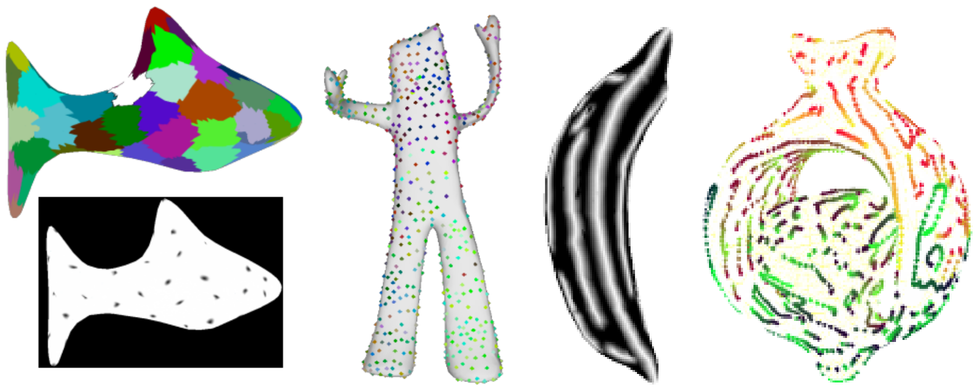


Fig. 10. From left to right: Equal area regions, distributing points across the surface, and lines that follow principle curvature directions.

7.4. *Principle curvature lines*

It has been demonstrated that lines placed along directions of principle curvature can help distinguish shape [22]. One problem with using curvature calculated directly from the mesh is that it tends to be noisy, making long lines difficult to draw or requiring substantial filtering. An alternative approach is to use an analytical surface approximation. To place lines along the direction of principle curvature we begin at a random point in the manifold and follow the appropriate parameter space vector (the curvature calculation computes the vectors in parameter space corresponding to the minimum and maximum tangent curvature vectors). We stop when we are too close (as measured on the surface) to an existing line or if the curvature vectors disappear. The parameter lines are stored as polylines in the domain.

We use an area calculation to ensure that the step size is a fraction of the desired curvature line spacing. To speed up the nearest neighbor calculations we use the n-ary partition described above to keep track of the distance to the closest line. As each new point is added to the polyline we simply update the distances stored in neighboring grid squares (until we are further away than the desired line spacing).

Principle curvature lines are shown in Figure 10. Note that no filtering was applied to the curvature calculations.

8. Conclusion and Future Work

We have presented example manifolds for use with all genus surfaces without boundary, and two surfaces with boundary. Given a mesh, we create a bijection between the mesh and the manifold of corresponding genus. We can then embed the manifold using any planar embedding technique, such as splines or radial basis functions. This embedded, analytical surface can then be used to generate surface information such as curvature, distance between two points on the surface, and surface area. It should be noted that the manifold embedding formulation provides a mechanism for converting any existing planar embedding approach to arbitrary topology surfaces without the need for additional geometric constraints.

The eventual goal of this work is to provide a disciplined mechanism for converting existing 2D, planar algorithms to surfaces of arbitrary topology. We are currently exploring surface segmentation and comparison algorithms based on existing 2D image-segmentation algorithms.

In the area of parameterization, a better, automated technique for guaranteeing a bijection for the $n-$holed tori is needed. Co-homology groups, combined with mesh decimation, is a promising approach. We are also exploring the incorporation of recently-introduced parameterization algorithms for non-zero genus meshes, as well as user-provided constraints.

Acknowledgements

This work was supported in part by NIH grant AR44005.

References

1. M. Alexa. Merging polyhedral shapes with scattered features. *The Visual Computer*, 16(1):26–37, 2000.
2. P. N. Azaraiadis and N. A. Aspragathos. Geodesic curvature preservation in surface flattening through constrained global optimization. *Computer-Aided Design (CAD))*, 33(8):581–591, 2001.
3. C. Bennis. Piecewise surface flattening for non-distorted texture mapping. *Computer Graphics*, 25(4):237–246, 1991.
4. H. Biermann, A. Levin, and D. Zorin. Piecewise smooth subdivision surfaces with normal control. *Proceedings of SIGGRAPH 2000*, pages 113–120, July 2000. ISBN 1-58113-208-5.
5. J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. *Proceedings of ACM SIGGRAPH 2001*, pages 67–76, 2001.
6. M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. *Eurographics)*, 21(2), 2002.
7. T. K. Dey and H. Schipper. A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection. *Discrete & Computational Geometry*, 14:93–110, 1995.
8. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH*, pages 173–182, 1996.
9. H. Ferguson and A. Rockwood. Multiperiodic functions for surface design. *Computer Aided Geometric Design*, 10(3):315–328, Aug. 1993.
10. M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997. ISSN 0167-8396.
11. M. S. Floater, K. Hormann, and M. Reimers. Parameterization of manifold triangulations. *Approximation Theory X: Abstract and Classical Analysis*, pages 197–209, 2002.
12. M. S. Floater and M. Reimers. Meshless parameterization for surface reconstruction. *Computer Aided Geometric Design*, 18(2):77–92, March 2001. ISSN 0167-8396.

13. D. Forsey and R. Bartels. Hierarchical b-spline refinement. *Computer Graphics*, 22(2):205–212, July 1988. Proceedings of SIGGRAPH '88.
14. C. Gotsman, X. Gu, and A. Sheffer. Fundamentals of spherical parameterization. In *Siggraph*, 2003.
15. C. Grimm. Simple manifolds for surface modeling and parameterization. *Shape Modelling International*, May 2002.
16. C. Grimm and J. Hughes. Modeling surfaces of arbitrary topology using manifolds. *Computer Graphics*, 29(2), July 1995. Proceedings of SIGGRAPH '95.
17. C. Grimm and J. Hughes. Smooth iso-surface approximation. *Implicit Surfaces*, pages 57–67, 1995.
18. C. Grimm and J. Hughes. Parameterizing n-holed tori. *Mathematics of Surfaces IX (to appear)*, 2003.
19. X. Gu and S.-T. Yau. Computing conformal structures of surfaces. *Communications in information and systems*, 2(2):121–145, 2002.
20. I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 95–102, July 2000.
21. S. Haker, S. Angenent, A. Tannenbaum, R. Kikinis, G. Sapiro, and M. Halle. Conformal surface parameterization for texture mapping. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):181–189, April - June 2000. ISSN 1077-2626.
22. V. L. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. *Computer Graphics*, 31(Annual Conference Series):109–116, 1997.
23. A. Khodakovsky, N. Litke, and P. Schröder. Globally smooth parameterizations with low distortion. In *Proceedings of ACM SIGGRAPH 2003*, Computer Graphics Proceedings, Annual Conference Series, July 2003.
24. V. Kraevoy, A. Sheffer, and C. Gotsmann. Constructing constrained texture maps. In *Siggraph*, 2003.
25. V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH*, pages 313–324, 1996.
26. A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98*, pages 95–104, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.
27. S. Lefschetz. *Introduction to Topology.* Princeton University Press, Princeton, New Jersey, 1949.
28. B. Lévy. Constrainted texture mapping for polygonal meshes. In *SIGGRAPH*, pages 417–424, 2001.
29. B. Lévy and J.-L. Mallet. Non-distorted texture mapping for sheared triangulated meshes. In *SIGGRAPH*, pages 343–352, 1998.
30. B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH*, 2002.
31. P. Lewis. Modeling surfaces of arbitrary topology with complex manifolds. Master's thesis, Brown university, 1996.
32. S. K. Lodha and R. Franke. Scattered data techniques for surfaces. *Scientific visualization (Proc. Dagsthuhl)*, pages 181–222, 1999.
33. J. Maillot, H. Yahia, and A. Verroust. Interactive texture mapping. *Proceedings of SIGGRAPH 93*, pages 27–34, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California.
34. M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential geometry operators for triangulated 2-manifolds. *VisMath*, jul 2002.

35. J. C. Navau and N. P. Garcia. Modeling surfaces from meshes of arbitrary topology. *Computer Aided Geometric Design*, 17(7):643–671, August 2000. ISSN 0167-8396.
36. J. C. Navau and N. P. Garcia. Modelling surfaces from planar irregular meshes. *Computer Aided Geometric Design*, 17(1):1–15, January 2000. ISSN 0167-8396.
37. H. K. Pedersen. Decorating implicit surfaces. In *SIGGRAPH*, pages 291–300, 1995.
38. D. Piponi and G. D. Borshukov. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 471–478. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, July 2000. ISBN 1-58113-208-5.
39. E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. *Proceedings of SIGGRAPH 2000*, pages 465–470, July 2000. ISBN 1-58113-208-5.
40. E. Praun and H. Hoppe. Spherical parameterization and remessing. In *SIGGRAPH*, 2003.
41. E. Praun, W. Sweldens, and P. Schröder. Consistent mesh parameterizations. *Proceedings of SIGGRAPH 2001*, pages 179–184, August 2001. ISBN 1-58113-292-1.
42. A. Sheffer, , and E. de Sturler. Angle based flattening of tesselated surfaces. In *SIAM Conference on geometric design and computing*, 2001.
43. A. Sheffer. Spanning tree seams for reducing parameterization distortion of triangulated surfaces. In *Shape Modeling and Applications*, pages 61–67, May 2002. ISBN 0-7695-1546-0, held in Banff, Canada.
44. A. Sheffer and J. Hart. Seamster: Inconpicuous low-distortion texture seam layout. In *IEEE Visualization (Vis02)*, pages 291–298, 2002.
45. A. Sheffer and E. D. Sturler. Smoothing an overlay grid to minimize linear distortion in texture mapping. *ACM Transactions on Graphics*, 21(4), 2002.
46. Y. Shinagawa, R. Kawamichi, T. Kunii, and S. Ohwada. Developing surfaces. *Shape Modelling International*, May 2002.
47. I. Singer and J. A. Thorpe. *Lecture Notes on Elementary Topology and Geometry.* Scott, Foresman and Company, Glenview, Illinois, 1967.
48. G. Turk. Generating textures for arbitrary surfaces using reaction-diffusion. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):289–298, July 1991. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.
49. A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics*, 28(Annual Conference Series):269–277, 1994.
50. G. Zigelman, R. Kimmel, and N. Kiryati. Texture mapping using surface flattening via multidimensional scaling. *Transactions on Visualization and Computer Graphics*, April 2002.
51. R. Zonenschein, J. Gornes, L. Velho, and N. Rodgriguez. Towards interactivity on texturing implicit surfaces: A distributed approach. In *Ninth international conference in central Europe on Computer Graphics, visualization and interactive digital media (WSCG 2001)*, 2001.