Simple Manifolds for Surface Modeling and Parameterization

Cindy M. Grimm
cmg@cs.wustl.edu

Abstract

We present a surface modeling technique using manifolds. Our approach uses a single, simple parameterization for all surfaces of a given genus. This differs from previous approaches which build a parameterization based on the elements of a mesh. The simple parameterization is more appropriate for applications that do complex operations in parameter space or on the mesh surface. We define a manifold and a corresponding embedding function for three genera (plane, sphere, and torus). The manifold can be used simply as a parameterization tool or as a smooth surface approximating the original mesh. We demonstrate how to build a correspondence between the mesh and the manifold, then how to build an embedding that approximates the mesh.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling, Curve, Surface, Solid, and Object Representations, Splines, texture mapping

1 Introduction

Manifolds are a technique for analyzing (or building) a surface by describing it as a collection of overlapping, simple surfaces. One key difference between manifolds and spline surfaces is that the simple surfaces *overlap*, instead of abutting, which makes it easier to move across the surface. One interesting aspect of manifolds is that the domain is specified separately from the embedding. This means we can create a manifold for an existing mesh and use it as a texture map without explicitly building an embedded surface.

Manifolds are a powerful technique for analysing surfaces [13][8] but their use as a modeling tool [5][10][11] has been limited. This is partly because the manifold surface constructions are somewhat unwieldy, lacking the simplicity of a technique such as subdivision surfaces. However, manifolds are increasingly useful in the problem of parameterization, in particular, texture mapping [9][12]. Here they are a natural, formal method for mapping rectangles of texture to a surface without introducing obvious seams. This paper describes a manifold with a simple structure, making it a useful tool for applications that need to perform complicated operations in parameter space or across a mesh surface.

The manifold can also be used as a surface model, which we demonstrate by fitting an embedded manifold to several different meshes. The manifold definition is, however, designed to simplify parameter space operations. The subsurfaces are few and simple (maps from uniform squares) and the transition functions between the sub-surfaces are easy to compute. To produce interesting surfaces we rely on a more complex embedding function built along the lines of hierarchical B-Splines [3]. We build one manifold per genus type, and use the same manifold for all surfaces of that genus. Geometric differences are created by using a different embedding for each surface.

We first discuss related work (Section 2). We then give the general format for defining manifolds, and the specific manifold definitions for a plane, sphere, and torus. In Section 4 we define an embedding function for the manifold that produces a surface based on splines. Section 5 describes how to establish a correspondence between a manifold and a mesh. Finally, we close with a discussion.

2 Previous work

There are three papers which describe a surface construction technique using manifolds [5][10][11]. Grimm's approach [5] begins with a mesh and builds a manifold with one chart per mesh element. The approach in Navau and Garcia's first paper [11] 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 [10] 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.

Texture mapping on non-planar surfaces requires a structure similar to manifolds, even if a manifold is not built explicitly. Several papers [15][16][14] build an ad-hoc structure that resembles a manifold but lacks a manifold's formal qualities. Two texture mapping papers [9][12] have explicitly built C^0 manifolds. Manifolds have also been used for for smoothing subdivision surfaces [1].

3 Manifold definition

The original definition of a manifold can be found in the topology literature [13] [8]. The basic idea is to analyze a complicated surface by defining maps[1] 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*. We can also define *transition functions* that take points between overlapping charts (see Figure 1).

The real-world analogy to a manifold is an atlas of the world. Each of the pages represent a mapping from the earth (a sphere) down to the plane. The pages overlap so that you can navigate from one page to an overlapping one (although in general the overlaps do not line up perfectly).

In Grimm [5] this definition was inverted in order to produce a manifold from a set of charts and transition functions. The essential part of this theorem is that the transition functions be reflexive, associative and transitive. The continuity of the manifold is the continuity of the transition functions. One thing to note here is that the *embedding* of the manifold is separate from the manifold definition itself; to embed the manifold each chart is embedded individually and the result blended together using a smooth partition of unity defined over the manifold.

In this paper we use a hybrid approach, combining the above two approaches to create an embedded manifold. We first create a manifold to serve as the domain of our surface by taking a canonical surface (*e.g.*, a sphere) and defining a small set of overlapping charts on it. The transition functions are then defined by mapping to and from the prototype surface. To define the final embedded surface, however, we use a version of the embedding function described by Grimm [5]. This approach creates a surface by embedding each chart and blending the results. This embedding function is described in Section 4.

We now introduce some definitions:

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

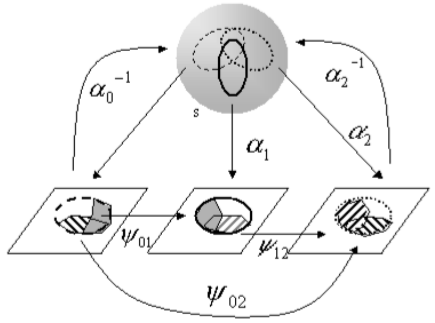

Figure 1. A surface S and three charts. Each chart maps a portion of the surface down to R^2. Transition function take points in charts to points in other charts. They must be consistent, *i.e.*, $\psi_{02} = \psi_{12} \circ \psi_{01}$.

- $\mathcal{S}$ is our canonical surface (one for each genus).
- A finite set, A, of maps from $\mathcal{S}$ to the unit square $((0,1) \times (0,1))$. 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. *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.
- 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. We build our Ψ functions, using the canonical surface, by defining ψ_{ij} to be $\alpha_i \circ \alpha_j^{-1}$.
- A point on the manifold can be defined in two ways. The first is as a point $p \in \mathcal{S} \subset \mathrm{R}^3$. We can also express p as a tuple of chart points, one tuple for each chart that contains the point p. A chart point is written as $[\alpha_c \in A, (x, y) \in c]$.

To build our manifolds we need to define the α_c functions. If these functions (and their inverses) are C^k then our manifold will be C^k. The structure is a manifold by the original definition [13]. We will now give the construction of the manifolds for each genus.

3.1 Plane

The plane manifold consists of a single chart defined on the uniform square.

3.2 Sphere

We use six copies of the standard sphere equation for our six charts, one at each pole (see Figure 2). Each chart covers

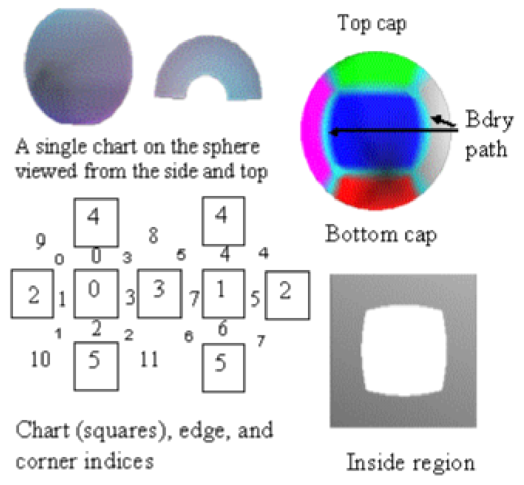
Top cap
A single chart on the sphere viewed from the side and top
Bdry path
Bottom cap
Chart (squares), edge, and corner indices
Inside region

Figure 2. Building charts for the sphere.

almost a half of the sphere.

$$\theta = u\pi, \qquad \phi = v\frac{3\pi}{4} - \frac{3\pi}{8}$$

$$\begin{aligned}
\alpha_0^{-1}(u,v) &= \big(\cos(\theta)\cos(\phi), \sin(\theta)\cos(\phi), \sin(\phi)\big)\\
\alpha_1^{-1}(u,v) &= \big(\cos(\theta+\pi)\cos(\phi), \sin(\theta+\pi)\cos(\phi), \sin(\phi)\big)\\
\alpha_2^{-1}(u,v) &= \big(\sin(\theta)\cos(\phi), \sin(\phi), \cos(\theta)\cos(\phi)\big)\\
\alpha_3^{-1}(u,v) &= \big(\sin(\theta+\pi)\cos(\phi), \sin(\phi), \cos(\theta+\pi)\cos(\phi)\big)\\
\alpha_4^{-1}(u,v) &= \big(\sin(\phi), \cos(\theta)\cos(\phi), \sin(\theta)\cos(\phi)\big)\\
\alpha_5^{-1}(u,v) &= \big(\sin(\phi), \cos(\theta+\pi)\cos(\phi), \sin(\theta+\pi)\cos(\phi)\big)
\end{aligned}$$

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)).

$$\begin{aligned}
\alpha_0(x,y,z) &= \big(\frac{\text{atan2}(y,x)}{\pi}, (\arcsin(z)+\frac{3\pi}{8})\frac{4}{3\pi}\big)\\
\alpha_1(x,y,z) &= \big(\frac{1+\text{atan2}(y,x)}{\pi}, (\arcsin(z)+\frac{3\pi}{8})\frac{4}{3\pi}\big)
\end{aligned}$$

The transition functions $\psi_{0,1}, \psi_{2,3}, \psi_{4,5}$ and their inverses are empty.

3.3 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 using the equation:

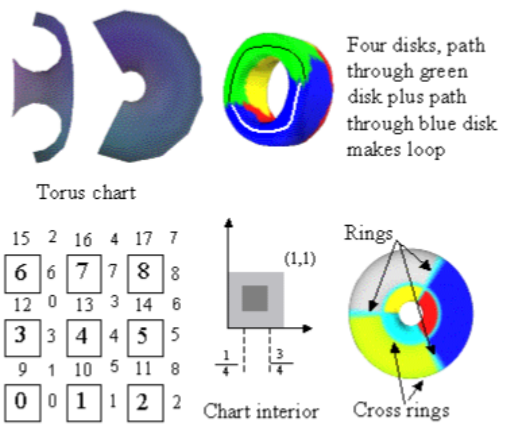
Four disks, path through green disk plus path through blue disk makes loop
Torus chart
Chart interior
(1,1)
Rings
Cross rings

Figure 3. Building charts for the torus.

$$T(\theta,\phi) = \big((1.5+\cos(\theta))\cos(\phi), (1.5+\sin(\theta))\cos(\phi), \sin(\theta)\big)$$

We use nine charts, each of which overlaps two-thirds of the torus function's domain. Numbering with chart zero in the lower left corner and two in the lower right corner we have:

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

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

$$\begin{aligned}
radius &= ||(x,y)|| - 1.5\\
\theta &= \text{atan2}(z, radius)\\
\phi &= \text{atan2}(y,x)\\
u &= \begin{cases} \theta/(2\pi) & \frac{\theta}{2\pi} < 0\\ \theta/(2\pi)+1 & otherwise \end{cases}\\
v &= \begin{cases} \phi/(2\pi) & \frac{\phi}{2\pi} < 0\\ \phi/(2\pi)+1 & otherwise \end{cases}\\
s &= \begin{cases} (u+1-\frac{(c \bmod 3)}{3}) & (c \bmod 3)=2, u<.5\\ (u-\frac{(c \bmod 3)}{3}) & otherwise \end{cases}\\
t &= \begin{cases} (v+1-\frac{(c/3)}{3}) & (c/3)=2, u<.5\\ (u-\frac{(c/3)}{3}) & otherwise \end{cases}
\end{aligned}$$

The torus transition functions are all translations by $\pm 1/3$.

4 Embedding

The manifold described in the previous section serves as the *domain* for either a mesh (if we are just parameterizing an existing mesh) or for a final surface (if we are using manifolds for surface modeling). The manifold is embedded by specifying an embedding for each chart, then blending the results together. To blend, we create a partition of unity on the manifold that says how much to take of each chart embedding at every point. The blend function is essentially a "bump" on each chart. Let $E_c : c \to \mathrm{R}^3$ be the C^k embedding function for a chart (typically a spline patch). Let $b_c : M \to \mathrm{R}$ be a function on the manifold M which is C^k continuous, and zero everywhere except over the chart c. The embedding equation is then:

$$E(p) : M \to \mathrm{R}^3 = \sum_c b_c(p) E_c(\alpha_c(p))$$

This equation is valid since b_c will be zero whenever E_c is undefined.

To build the b_c functions we use proto-functions defined on the chart, then promote them to functions on the manifold by defining them to be zero elsewhere. Let $\hat{b}_c : c \to \mathrm{R}$ be a function which is C^k continuous, positive over the chart, goes to zero by the boundary of the chart, and has all k derivatives zero at the boundary as well. (We use a single spline basis function whose support is the chart.) Since the derivatives go to zero by the boundary of the chart, when we promote the function to a function on the chart it will still be C^k. The blend functions are then:

$$b_c(p) : M \to \mathrm{R} = \frac{\hat{b}_c(\alpha_c(p))}{\sum_c \hat{b}_c(\alpha_c p)}$$

where $\hat{b}_c$ is defined to be zero when α_c is invalid. By dividing by the sum we produce a partition of unity, provided the denominator is never zero. By definition our $\hat{b}$ functions have positive support over the entire chart so we easily ensure the denominator is never zero.

All of our charts are defined to be the uniform square ($(0, 1) \times (0, 1)$). The embedding function for a given chart is based loosely on hierarchical B-splines [3]. Because we are using so few charts, the embedding function for a given chart may need to be fairly complicated. We use a single B-spline that covers the entire chart, plus subsequent spline surfaces that act as offsets.

$$E_c(u, v) = S_0(u, v) + \sum_1^n S_i(u, v)$$

We differ from hierarchical B-splines in one respect; instead of constraining the derivatives of the S_i patch to ensure continuity, we simply evaluate S_i on all of the uniform square, not just the traditional support area where the basis functions sum to one. We discuss fitting to a mesh in Section 5.6.

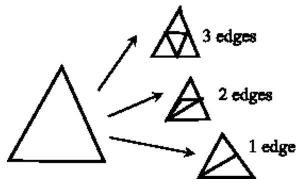

Figure 4. Splitting a triangle. If the face or all of its neighbors need to be split, we split into four triangles. If two of the faces neighbors need to be split we produce three triangles. If only one neighbor is splitting we produce two triangles.

4.1 Tessellation

To tessellate the surface we first tessellate the interior of each chart, sharing edges and vertices according to the overlap information. We then further split poorly fitting faces into four smaller triangles (see Figure 4) until each face has at most n mesh vertices mapping to it, for some small n. This produces an adaptive triangulation.

The interior region is chosen so that the union of the regions exactly covers the surface and none of the regions overlap (refer to Figures 2 and 3). For the plane this is just a tessellation of the single chart. For the sphere this is approximately the region given by $[0.25, 0.75] \times [0.25, 0.75]$. For the torus this is exactly $[0.25, 0.75] \times [0.25, 0.75]$.

For the sphere the interior boundaries do not lie along straight lines. We chose the interior boundary points by averaging between two straight lines (at s or $t = 0.25$ or 0.75), one in each chart. We map both boundary points to the sphere, average their location on the sphere, then map back into one of the two charts. This produces the interior region shown in Figure 2.

5 Fitting to a mesh

Once we have defined our canonical manifolds we next need to establish a correspondence between the appropriate genus manifold and an input mesh. After establishing this correspondence we can construct an embedding for the manifold that approximates the mesh.

This section is broken into the following subsections: A definition of the correspondence, general mesh operations used in building the correspondence, a specific algorithm for each genus, and finally, constructing an embedding that approximates the mesh.

5.1 Mesh to manifold correspondence

Let M be the manifold and $\{V, E, F\}$ be the vertices, edges, and faces of the mesh. We wish to define a function $\mathcal{M} : M \rightarrow$ mesh which is one-to-one, and an inverse one-to-one function which takes the mesh to the manifold. We define $\mathcal{M}$ by first defining it at the vertices, then extending it to the faces using barycentric coordinates[2]. We construct a mapping for the vertices such that there exists at least one chart which contains all of the mapped vertices for each face. This creates a mapping that takes each face of the mesh to a triangle in one or more charts. Using barycentric coordinates we can define a 1-1 and onto correspondence between the mesh face and the corresponding chart triangle. Note that the triangles in different charts may encompass a different subset of the manifold if the transition function is not affine.

We first define some terminology. Let v_i^f be the vertices of a face f. Any point in f can be defined using barycentric coordinates, *i.e.,* $p = \sum \beta_i v_i^f$ where the β_i are positive and sum to one. To define a corresponding triangle in a chart c we use the three points $v_i^{f_c} = \mathcal{M}^{-1}(v_i^f)$. A point in the chart triangle can also be defined using barycentric coordinates β_i^c. This gives us a 1-1 and onto function from a chart to a subset of the mesh faces:

$$\begin{aligned}\mathcal{M}_c((u,v) \in f_c) &= \sum \beta_i^c \mathcal{M}(v_i^{f_c}) \\ \mathcal{M}_c^{-1}(p \in f) &= \sum \beta_i \mathcal{M}^{-1}(v_i^f)\end{aligned}$$

If the transition functions are affine (as in the case for the torus and plane) then this function can be extended to the entire manifold by just picking a chart for each face. If not then we chose a chart based on which chart has the projected face closest to the center of the chart. This will produce minor inconsistencies between $\mathcal{M}$ and its inverse along boundaries between faces that have different default charts.

For the above approach to work we need to define a vertex correspondence such that every face in the mesh has a chart it maps into. In addition, we require the following:

- The number of vertices assigned to each chart is approximately equal.
- There exists a chart such that $\mathcal{M}(v)$ and $\mathcal{M}(v^*)$ (the star of v, i.e., all of v's neighbors) lie in that chart.
- The polygon formed by mapping v^* into a chart contains the mapped point for v.
- Ideally, the above polygon should be convex.

Our general approach is to partition the mesh into n disk regions, assigning each of these disks to the interior of a chart. This produces an initial projection of the vertices of the mesh to the charts, which we then refine to meet the above criteria. Partitioning requires several basic mesh operations, which we describe below.

[2]If the mesh's faces are not triangular then we first triangulate.

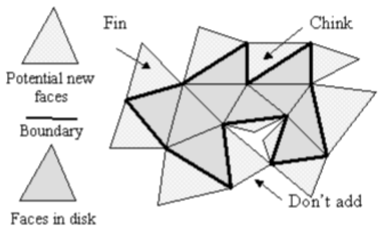

Figure 5. A disk with its current boundary. All faces that share a boundary edge are potential faces to add. We do not add faces whose boundary is not contiguous along the disk boundary. When the disk is finished we can optionally remove "fins" and "chinks" by removing or adding the marked face.

The method described here is appropriate for meshes of 400 to 3,000 vertices. If the mesh is too small there is not enough space to run the partition boundary edges; in this case we can apply subdivision to introduce new vertices. If the mesh is too big, we can apply progressive meshes, fit the smaller mesh, then add the vertices back in [6][7].

5.2 General partitioning tools

We describe several (standard) mesh algorithms needed for partitioning the mesh. In the remainder of this section *distance* refers to distance on the mesh, *i.e.,* the number of edges traversed, not Euclidean distance.

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 run an additional routine that takes out any chinks or "fins" in the boundary (see Figure 5).

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 in the chart for the boundary vertices, find locations for the interior disk vertices. We use Floater's algorithm [2] which provides a least-squares solution. There is one linear equation for each boundary vertex ($\mathcal{M}^{-1}(v) = (u,v)$) and one equation for each interior vertex ($\mathcal{M}^{-1}(v) = 1/n \sum \mathcal{M}^{-1}(v^*)$) that places each vertex at the centroid of its neighbors.

Reproject Given $\mathcal{M}^{-1}(v)$ for all vertices, reproject the vertices of a chart. First, find the largest disk in the mesh that contains only vertices that map into that chart. Second, move the locations of the boundary vertices in towards (or out from) the center $(0.5, 0.5)$ of the chart by an amount proportional to the number of faces in this chart divided by the average number of faces per chart. Call the **Project** routine.

Adjust centroids Given $\mathcal{M}^{-1}(v)$ for all vertices, move each vertex towards the centroid of its neighbors.

5.3 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 Sphere

Fitting the sphere takes two steps. We first partition the mesh into six disk regions, each with four boundary edges (see Figure 2). We project these disks onto the interior of each chart, placing the vertices from the boundary edges along the interior edges defined in Section 4.1. We then iterate, reprojecting and adjusting centroids, until the criteria given above are met.

To make the six regions we first make the two end cap regions, then join them together with four boundary paths to create the other four boundary regions. The caps are made by running two **Grow disk** routines at the same time, alternately adding rings. The **Grow disk** routine is seeded with two faces that are far apart in the mesh.

The four boundary paths are made by joining four vertices spaced evenly around the boundaries of both disks. We run **Shortest path** between these four pairs to create the boundary paths. To make sure these paths do not cross each other, we mark the path vertices as inaccessible as the paths are created. Picking the evenly spaced boundary points can be difficult because not all of the vertices on the boundary are accessible. We limit our search for the four boundary vertices to those that are accessible.

Once we have the boundary paths we flood fill the regions between them to produce our remaining four disks. The boundary paths form the interior boundaries for the **Project** call.

We now iterate, reprojecting the interior of each chart using the **Reproject** routine, and adjusting centroids. We find reprojecting the smallest, then the largest, then all of the charts, followed by 30 or so iterations of adjusting the centroids works best. If all vertices lie in the polygon formed by their neighbors and the maximum variance in number of faces is less than 0.2, stop.

We have no proof that this method converges, or that there even exists a solution that meets all of the criteria given above, for all meshes. If the surface that the mesh approximates is topologically a sphere then there must be six disks (of possibly unequal size) that adjoin correctly. However, the surface may not be sampled sufficiently (*i.e.*, there may not be enough vertices) for the vertices to lie flat in the charts. In particular, if a mesh has a skinny tube, like Gumby's arms (see Figure 6), and there are not enough vertices around the arm to be able to "flatten" the arm nicely into a disk, then this will fail.

5.5 Torus

For the torus we need to find six rings, three that loop one way around the torus, and three that loop perpendicular to the first three rings (see Figure 3). This will partition the mesh into nine correctly adjacent disks. Then, like the sphere, we can iterate, reprojecting, until the projection meets the desired criteria.

The first ring must be a loop that cannot be contracted into a single vertex [8]. 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 exactly two 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.

Once we have a single loop we can create the other two using the **Grow disk** routine. We seed one **Grow disk** routine with faces from the loop oriented one way, and one with the loop oriented the opposite way. This actually grows two annuli which will meet in another loop. To get a loop that is one third of the way around, we grow one annulus at twice the speed of the other. Once we have the second loop, we again grow two annuli from the two loops, creating the third loop when they meet.

The parallel loops are made in a fashion similar to the four boundary paths in the sphere. We find three triplets of points, (three points per loop) and run **Shortest path** three times to create a single loop. Again, care must be taken to only consider accessible boundary points.

5.6 Fitting the embedding

We first discuss error calculation, then how we decide where to add more patches.

5.6.1 Error calculation

We calculate the error for a given point (u, v) in a chart by

$$\Delta(u,v) = \frac{b_c(u,v)}{||(b_c(0.5,0.5)||}\big(\mathcal{M}(u,v) - E_c(u,v)\big)$$

We scale by the blend function to avoid over-fitting at the boundary of the chart where the contribution of E_c is small. Calculating $\mathcal{M}$ involves searching through all of the polygons that cover the chart to find the one that encloses the given point. To speed this up, we divide the chart into even-sized bins and use a scan-conversion routine to determine which polygons lie in which bins. This limits the number of polygons we need to search.

5.6.2 Adding spline patches

Recall that the embedding for a given chart consists of one base spline patch plus n spline patches which are treated as offsets. Since we are blending the individual patch embedding together for the final surface it suffices to fit each chart individually. We also have a one-to-one correspondence between the manifold and the mesh. We use the least-squares fitting technique [4] to fit the patches. This leaves us with two questions; how many patches and what their supports should be, and which pin-points to use for the least-squares fit.

To determine the number of patches we alternate between fitting the current set and adding new patches where the fit is bad. We search for the largest rectangle that covers a connected bad spot and add a patch whose support covers that rectangle. We stop when the average and maximum error for all vertices is below given thresholds.

To find the "bad" rectangle we divide the chart into n by n bins, choosing n so that the bin size is less than four times the smallest projected mesh edge *i.e.*, $\min ||\mathcal{M}(e_{v0}) - \mathcal{M}(e_{v1})||$. To determine the fit of a bin we calculate Δ at the center of the bin. In addition, we take each vertex and measure $\Delta(\mathcal{M}(v_p))$, flagging the bin v projects into as bad if the fit is bad. We then grow connected areas of bad bins and pick the largest.

To produce the pin points we use both the vertices of the mesh and evenly spaced points in the domain (corresponding points on the mesh can be found using the $\mathcal{M}^{-1}$ function). We need evenly spaced points to ensure that the least-squares fitting of the spline patch is well-behaved.

6 Results

We demonstrate the algorithm on a number of spherical and toroidal meshes. The meshes are colored by chart to give an indication of how they were partitioned. For the toroidal mesh and gumby we embedded the mesh using the $\mathcal{M}$ function. For the fish and banana we created an embedding with C^2 spline patches with 7×7 control points. The banana mesh had 504 vertices and required between 2 and 4 spline patches per chart. The fish had 1296 vertices and required between 2 and 5 patches per chart (see Figure 6).

7 Conclusion

Manifolds are a promising technique for parameter-space operations. We have described a simple manifold for three basic genera and show that it can be used to parameterize several sample meshes. This is the first step towards creating a general-purpose, easy to use tool for mesh parameterizing.

References

[1] Henning Biermann, Adi Levin, and Denis Zorin. Piecewise smooth subdivision surfaces with normal control. *Proceedings of SIGGRAPH 2000*, pages 113–120, July 2000. ISBN 1-58113-208-5.

[2] Michael S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997. ISSN 0167-8396.

[3] D. Forsey and R. Bartels. Hierarchical b-spline refinement. *Computer Graphics*, 22(2):205–212, July 1988. Proceedings of SIGGRAPH '88.

[4] B. Fowler and R. H. Bartels. Constraint based curve manipulation. *Siggraph course notes 25*, July 1991.

[5] Cindy Grimm and John Hughes. Modeling surfaces of arbitrary topology using manifolds. *Computer Graphics*, 29(2), July 1995. Proceedings of SIGGRAPH '95.

[6] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH 99*, pages 325–334, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[7] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David 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.

[8] S. Lefschetz. *Introduction to Topology*. Princeton University Press, Princeton, New Jersey, 1949.

[9] Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. *Proceedings of SIGGRAPH 93*, pages 27–34, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California.

[10] J. Cotrina Navau and N. Pla Garcia. Modeling surfaces from meshes of arbitrary topology. *Computer Aided Geometric Design*, 17(7):643–671, August 2000. ISSN 0167-8396.

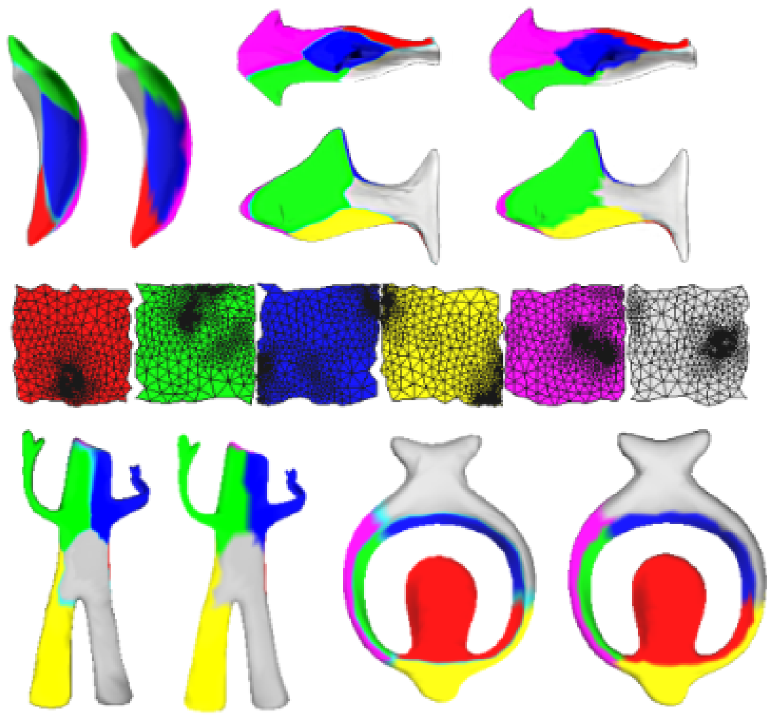

Figure 6. Fitting results. Right is original mesh. Meshes are colored by their chart centers. In the middle is the mapping of the fish mesh vertices to the charts.

[11] J. Cotrina Navau and N. Pla Garcia. Modelling surfaces from planar irregular meshes. *Computer Aided Geometric Design*, 17(1):1–15, January 2000. ISSN 0167-8396.

[12] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. *Proceedings of SIGGRAPH 2000*, pages 465–470, July 2000. ISBN 1-58113-208-5.

[13] I.M. Singer and J. A. Thorpe. *Lecture Notes on Elementary Topology and Geometry*. Scott, Foresman and Company, Glenview, Illinois, 1967.

[14] Karan Singh and Evangelos Kokkevis. Skinning characters using surface oriented free-form deformations. *Graphics Interface*, pages 35–42, 2000. ISBN 1-55860-632-7.

[15] Greg Turk. Texture synthesis on surfaces. *Proceedings of SIGGRAPH 2001*, pages 347–354, August 2001. ISBN 1-58113-292-1.

[16] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. *Proceedings of SIGGRAPH 2001*, pages 355–360, August 2001. ISBN 1-58113-292-1.