Consistent Spherical Parameterization

Arul Asirvatham[1], Emil Praun[1], Hugues Hoppe[2]

[1]School of Computing - University of Utah, USA
{arul, emilp}@cs.utah.edu,
[2]Microsoft Research, Redmond, USA
http://research.microsoft.com/~hoppe

Abstract.

Many applications benefit from surface parameterization, including texture mapping, morphing, remeshing, compression, object recognition, and detail transfer, because processing is easier on the domain than on the original irregular mesh. We present a method for simultaneously parameterizing several genus-0 meshes possibly with boundaries onto a common spherical domain, while ensuring that corresponding user-highlighted features on each of the meshes map to the same domain locations. We obtain visually smooth parameterizations without any cuts, and the constraints enable us to directly associate semantically important features such as animal limbs or facial detail. Our method is robust and works well with either sparse or dense sets of constraints.

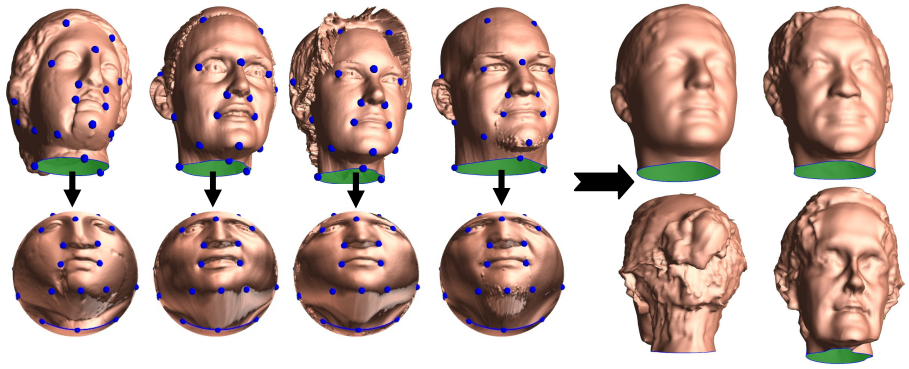

Fig 1. Consistent Spherical Parameterization of a collection of heads

1 Introduction

Several applications in computer graphics, such as texture mapping, compression, surface processing, detail synthesis, and object recognition rely on mesh parameterization. Parameterization refers to computing mappings between surfaces in 3D and simpler domains such as planar regions, simplicial domains, or spheres. To be useful for these varied applications, the mappings must satisfy several properties, such as

continuity, bijectivity, smoothness, constraint satisfaction, and acceptable distribution of domain area over the surface.

Many parametrization techniques involve partitioning the surface into simpler pieces using cuts. Such cuts are problematic for applications involving a large number of models, since they can lead to fragmentation of the map (i.e. smaller and smaller pieces across which there is a continuous map between all models). Moreover, the resulting map generally lacks smoothness along cuts and at their junctions.

Continuous parameterizations are only possible between topologically equivalent models. For the genus-zero models that we consider, the unit sphere is a natural parameterization domain since it is inherently smooth. Allowing the user to specify constraints is another important requirement in many applications, since it provides a way to incorporate higher-level semantic knowledge about the objects.

We extend the work of Praun and Hoppe [14] on spherical parameterization to allow simultaneous parameterization of multiple objects with point feature constraints while guaranteeing continuity, bijectivity, visual smoothness, and minimizing overall distortion. Using these consistent spherical parameterizations, one can create consistent geometry images [6; 14] representing several objects in a database, opening the door to a large array of applications including compression, conversion to hardware-supported subdivision surfaces with displacement maps, natural LODs, etc.

We present the following contributions:

- Robust construction of consistent spherical parameterizations for several surfaces.
- Constrained spherical parameterization where specified points on a mesh map to given points on the sphere.
- Methods to avoid swirls, and to correct them when they arise.

2 Previous Work

Planar parameterization: The earliest parameterization methods established mappings to planar domains. There have been many methods developed to date; for a survey we refer the reader to Floater and Hormann [3]. Unless applied to topological discs, these methods must cut the mesh into one or several pieces, introducing discontinuities in the parameterization.

Spherical parameterization: Discontinuities can be avoided altogether by mapping models to smooth domains of the same topology, such as the unit sphere for genus-zero models. Examples of spherical parameterization methods include [18; 1; 9;7; 5; 14]. We build upon the method of Praun and Hoppe [14].

Incorporating hard constraints into a spherical parameterization scheme proves to be challenging, because it is difficult to guarantee bijectivity of the result. We chose to build upon the approach of Praun and Hoppe [14] because its hierarchical construction approach enables a robust solution.

Parameterization constraints. Several methods address the problem of parameterization under constraints. Lévy [13] texture maps meshes under soft constraints by introducing additional terms in the quality metric. Eckstein et al. [2] propose a method for satisfying hard constraints for planar parameterization.

Kraevoy et al. [11] start with an unconstrained planar parameterization and then move the constrained vertices to their required positions by matching a triangulation

of these positions to a triangulation of the planar mesh formed by the paths between constrained vertices. Finally, they relax this parameterization while keeping the constraints. They demonstrate results involving a fairly large number of features.

In contrast, our method works well even with few provided feature points. In addition, we use a multiresolution approach to avoid forming an initial parameterization and then optimizing it.

Consistent parameterization. Praun et al. [15] consistently parameterize a set of genus-0 models onto a user-specified simplicial complex. Unlike their method, we do not impose a fixed consistent connectivity of the base domain, but only satisfy the given point constraints, thereby providing the map more freedom.

Two recent works, by Schreiner et al. [17] and Kraevoy and Sheffer [12], improve upon the technique of Praun et al. by not requiring the simplicial complex to be specified a priori. However, these new techniques do not scale well with regard to the number of models to be consistently parameterized. The technique of Schreiner et al. is limited to dealing with only 2 models, and while Kraevoy and Sheffer demonstrate consistent parameterization among 3 models, their approach is asymmetric and would not scale to a large collection of models.

Whereas in all previous cases the simplicial domain is abstract (with no inherent geometry), our spherical domain geometry is explicit. This implies that paths must be straight arcs on the sphere, rather than arbitrary meandering mesh paths, which makes the problem more difficult. One advantage of the lack of domain connectivity is the opportunity to improve the map by flipping edges, like in Delaunay refinement.

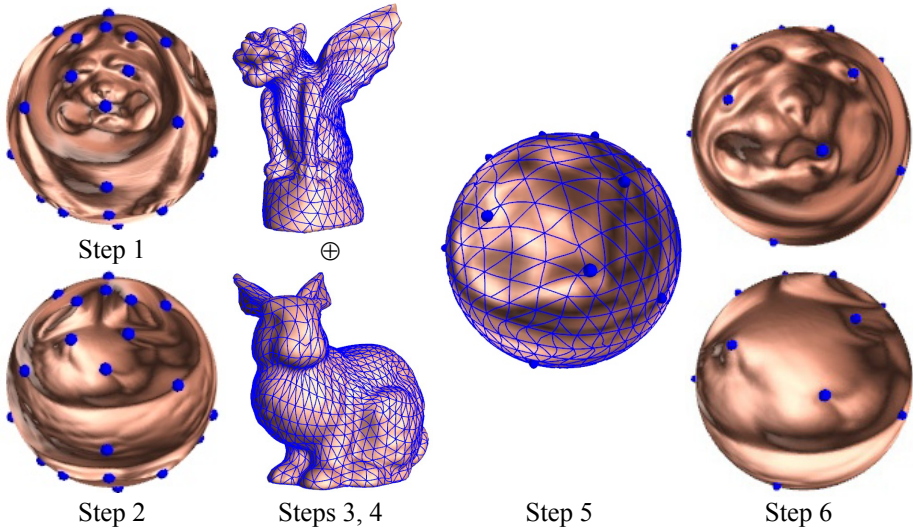

Fig. 2. Steps of the algorithm. For clarity, steps 3-5 show a coarser version of the remesh grid than the one actually used. The features in step 5 are vertices of the finer grid. Stretch efficiencies: gargoyle 0.632, bunny 0.697.

3 Approach

Given a set of meshes and corresponding feature points, we form a consistent parameterization of all the meshes. Our approach has two major parts. First, we find good spherical feature locations, such that the final maps have low distortion and distribute the sphere area adequately to the various parts of the input meshes. Second, we create a constrained spherical parameterization for each surface, forcing the feature points to map to the computed locations. Here is the algorithm in more detail (see also Figure 2). Let M_i (i=1..n) be the initial meshes, P be a parameterization, and F a set of spherical feature locations.

1. (P_1' , F') := UnconstrainedSphericalParam(M_1)
   //Initial feature locations F' on sphere using one model.
2. For i=2..n, P_i' := ConstrainedSphericalParam(M_i , F')
   //Parameterize all models using those initial locations.
3. For i=1..n, M_i^R := Remesh(M_i , P_i')
   //Remesh to n geometry images with identical connectivities.
4. M_*^R := { M_1^R , M_2^R , … , M_n^R }
   //Concatenate to single mesh with vertex coordinates in R^{3n}.
5. (P^R,F) := UnconstrainedSphericalParam(M_*^R)
   //Find good feature locations considering all models.
6. For i=1..n, P_i := ConstrainedSphericalParam(M_i , F)
   //Compute final parameterizations using these locations.

The main new procedure is the constrained spherical parameterization used in steps 2 and 6. It is described in detail in Section 4. The remaining steps are adapted from earlier work [14]. In step 3 we need to exactly represent the feature points, so we snap the closest grid samples to the spherical locations F' .

Step 5 involves the construction of a progressive mesh for the special mesh M_*^R with geometry in R^{3n}. We modify the quadric error metric [4] to sum each of the n errors in R^3. We also modify the spherical parameterization [14] to sum the n stretch energies from the sphere to the n mesh geometries in R^3.

4 Constrained Spherical Parameterization

We adapt the method of [14] to work with constraints as follows. During coarse-to-fine refinement, we can simply fix the spherical location of feature vertices. The difficulty is to bootstrap the algorithm by creating a valid starting state that satisfies all constraints. Specifically, we must create a progressive mesh representation of the surface where the base domain contains only feature vertices and is triangulated the same way as the spherical features (see Figure 3).

To satisfy this, we need only find a spherical triangulation and a corresponding embedding of its arcs onto the mesh, given by a set of non-intersecting paths. Once we have such a path network, we simplify the mesh to produce a progressive mesh, but we keep the feature vertices, and only allow vertices on the feature paths to be collapsed into other path or feature vertices [16].

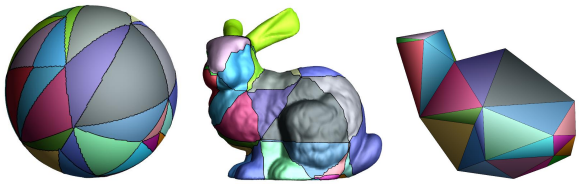

Fig. 3. "Triangulations" of features on the sphere (left) and mesh (middle); base mesh after simplification (right).

To produce the path network on the 3D surface, we use a method similar to those of Praun et al. [15] and Kraevoy et al. [11]. We link together pairs of feature points, with great circle arcs on the sphere, and paths on the mesh, until we complete a full "triangulation". The paths (and arcs) cannot intersect each other except at feature vertices (and their spherical locations). The algorithm proceeds in greedy fashion, selecting the best pair from a pool of candidates. To guarantee both termination and topological equivalence of the two "triangulations", we maintain consistent ordering of neighbors around each vertex in the partially completed graph, and we avoid adding any arcs/paths closing cycles before we have linked all the vertices in a spanning tree [15].

The candidate pool is populated initially by shortest paths between all the feature pairs, computed using a Dijkstra search on mesh vertices. When we select the best candidate path we check to see if it intersects any paths already inserted in the network and if it does we re-compute it using a restricted search. These restricted searches can also use edge midpoints in addition to mesh vertices (though with a cost penalty), corresponding to inserting "Steiner" vertices in the original mesh [11].

To improve the geometric quality of the triangulation we employ a set of heuristics to avoid and fix swirls (Section 4.1) and we flip edges by replacing them with the other diagonal of the quad formed by the two adjacent patches. This is done only when the new path is shorter and the new configuration is valid on the sphere.

4.1 Dealing with swirls

Sometimes the paths appear bad to a human observer because they take unnecessarily long routes around other feature vertices. We call a *swirl* the local configuration of these long paths (Figure 4).

Praun et al. [15] note that swirls cannot be repaired using 1-ring relaxations of patches around a vertex, and propose several heuristics to prevent them. Unfortunately we do not benefit from a user-provided base domain, so some of their heuristics do not work in our case. Furthermore, our setting is harder since the spherical locations of the feature points obtained in steps 1 and 5 of the algorithm (Section 3) may be quite different from preferred spherical locations considering the geometry of the

current mesh. We develop a more robust set of swirl-avoiding heuristics, as well as a method to remove swirls *after* they have appeared.

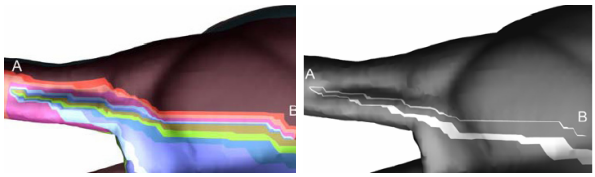

Fig. 4. Swirl on the horse leg: the white patch (right) has to connect to B, but it does so around A. It cannot be straightened sine it would have to move over A and AB.

Swirl-avoiding heuristics: The main tool we use in selecting new paths to insert in the network is their ranking in the priority queue. Paths are ranked according to their surface length (shorter is better), but they are occasionally penalized (placed at the end of the queue) when certain conditions occur. Such conditions include:

- The current path links non-extreme vertices, and there are still some extremities left unconnected. Extremities are features with large average distance to their nearest neighbors (such as legs, arms, etc.). We start the spanning tree construction by linking such features. If left unconnected they might cause swirls since paths linking other vertices go around the base of the extremity, equally likely on the "correct" as on the "wrong" side.
- The spherical image of the path would create spherical triangles with very small angles (<10 degrees in our examples). In these cases the winding order of the 3 feature locations on the sphere is not reliable. Furthermore, skinny triangles make the spherical optimization less robust.
- Failed sidedness tests for neighboring features. We check whether the projection of a feature vertex onto the path is on the same side as the projection of the corresponding feature point onto the arc on the sphere. If some of the vertices are on different sides on the mesh and the sphere, we try to force the path to lie on the correct side of nearby feature vertices. To do this we add temporary constraint paths from the path endpoints to the neighboring feature. We now trace the shortest path on the mesh. The path so traced will be on the correct side of all the features in the connected component of the neighbor. However, this might not always be possible as the addition of the temporary constraint paths might form a cycle enclosing either the source or the destination vertex on different sides on the mesh and the sphere. In such cases, we add the path to the end of the queue.

Unswirl operator. In addition to using heuristics that avoid swirls (as Praun et al. did [15]), we have also developed a method to identify and remove them in the rare cases that they do occur. Paths between two features incident to many other "long" paths (with high ratios between actual length and geodesic distance between endpoints) are likely to be the center of swirls. To fix them, we remove all paths incident to the two vertices, and then replace them in a new order, introducing first paths that were previously bad.

5 Results and Applications

Figure 1 illustrates the basic approach for applications making use of large model databases. To create the database, a few representative models are selected and con-

sistently parameterized using our algorithm, in order to obtain good spherical locations for the 22 feature points. In our example we used the 5 heads shown out of a set of 8. The remaining models can be subsequently added to the database by running *constrained* spherical parameterization. Once the database is created, the consistent parameterization can be used for tasks such as classification and retrieval based on principal component analysis. On the right side of Figure 1 we show the average of our set of heads, and the first three principal components (visualized added to the average head).

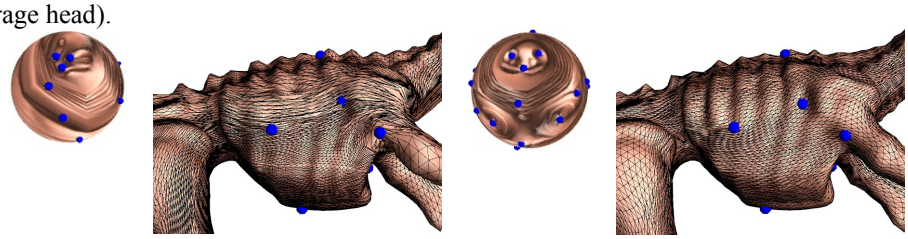

Fig. 5. Parameterization quality improves after optimizing the map taking into account all models.

Figure 5 demonstrates the role of steps 3-6 of the algorithm. The two left images show the spherical parameterization and regular remesh using the locations obtained using the cow model alone. The quality improves (right) when the feature locations are computed using all the models.

Consistent parameterizations can also be used to transfer mesh properties, such as geometric detail (the high-frequency components of a multiresolution mesh representation), normals, colors, or texture coordinates. Figure 6 demonstrates examples of color and normals transfer between two heads.

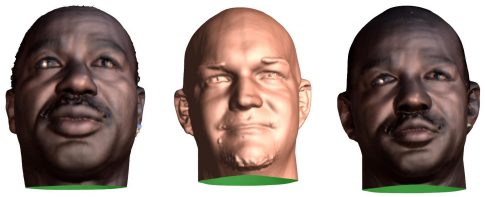

Fig. 6. The texture and normals of the head on the left are combined with the geometry of the head in the middle to produce the one on the right.

Table 1 shows timing results for our method. For small meshes, step 5 is the most expensive since it involves several geometries simultaneously. However, when the original meshes are dense, step 6 becomes the most expensive one.

Table 1 Timing results (in minutes) for the heads example (Figure 1), bunny and gargoyle example (Figure 2). The timings for steps 2 and 6 are cumulative for the different models in the set**.**

	Steps				
Models	1	2	5	6	Total time
Fig. 1	19	81	8	95	203
Fig. 2	10	5	5	17	37

6 Summary

We have presented a robust algorithm for parameterizing genus-zero models onto a sphere in the presence of feature constraints. The central part of the algorithm, *constrained* spherical parameterization is guaranteed to produce topologically equivalent

spherical triangulations and mesh patch partitions, and avoids awkward *swirl* configurations through a collection of novel heuristics. The regularly sampled consistent geometry images that can be obtained using our parameterizations allow digital geometry processing applicable to many real-world applications.

References

1. Alexa, M. 2000. Merging polyhedral shapes with scattered features. *The Visual Computer*, 16(1), pp. 26-37.
2. Eckstein, I., Surazhsky, V. and Gotsman, C. 2001. Texture mapping with hard constraints. *Eurographics 2001*, pp. 95-104.
3. Floater, M. and Hormann K. 2004. Recent advances in surface parameterization. *Multiresolution in geometric modeling 2004*.
4. Garland, M. and Heckbert, P. 1997. Surface simplification using quadric error metrics. *ACM SIGGRAPH 97,* pp. 209-216.
5. Gotsman, C., Gu, X. and Sheffer, A. 2003. Fundamentals of spherical parameterization for 3D meshes. *SIGGRAPH 2003,* pp. 358-364.
6. Gu, X., Gortler, S., And Hoppe, H. 2002. Geometry images. *ACM SIGGRAPH 2002,* pp. 356-361.
7. Gu, X., Wang, Y., Chan, T., Thompson, P, and Yau, S.-T. 2003. Genus zero surface conformal mapping and its application to brain surface mapping., *Information Processing Medical Imaging 2003*.
8. Guskov, I., Vidimče, K., Sweldens, W., and Schröder, P. 2000. Normal meshes. *ACM SIGGRAPH 2000,* pp. 95-102.
9. HAKER, S., ANGENENT, S., TANNENBAUM, S., KIKINIS, R., SAPIRO, G., AND HALLE, M. 2000. Conformal surface parameterization for texture mapping. *IEEE TVCG*, 6(2), pp. 181-189.
10. Hoppe, H. 1996. Progressive meshes. *ACM SIGGRAPH 96*, pp. 99-108.
11. Kraevoy, V., Sheffer, A. and Gotsman, C. 2003. Matchmaker: constructing constrained texture maps. *SIGGRAPH 2003,* pp. 326-333.
12. Kraevoy, V., and Sheffer, A. 2004. Cross-parameterization and compatible remeshing of 3D models. *SIGGRAPH 2004,* to appear.
13. Lévy, B. 2001. Constrained texture mapping for polygonal meshes. *ACM SIGGRAPH 2001*, pp. 417-424.
14. Praun, E. and Hoppe, H. 2003. Spherical parameterization and remeshing. *ACM SIGGRAPH 2003,* pp. 340-350.
15. Praun, E., Sweldens, W. and Schröder, P. 2001. Consistent mesh parameterizations. *ACM SIGGRAPH 2001,* pp. 179-184.
16. Sander, P., Snyder, J., Gortler, S., and Hoppe, H. 2001. Texture mapping progressive meshes. *ACM SIGGRAPH 2001*, pp. 409-416.
17. Schreiner, J., Asirvatham, A., Praun, E., and Hoppe, H. 2004. Inter-surface mapping. *ACM SIGGRAPH 2004*.
18. SHAPIRO, A. AND TAL, A. 1998. Polygon realization for shape transformation. *The Visual Computer*, 14 (8-9), pp. 429-444.