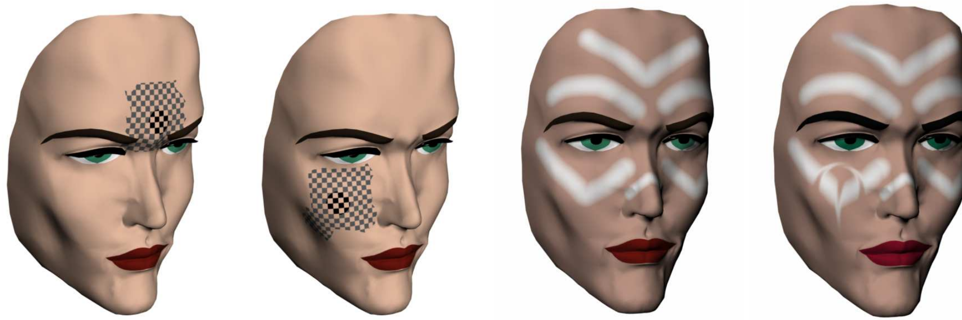
Volume xx (200y), Number z, pp. 1–8

Continuous Local Parameterization of Polygons

Jérôme Maillot †



Abstract

Texture mapping has now become a standard technique for adding visual details to 3d models. Nevertheless, intuitive mapping tools have made little progress in the past decade, compared to other computer graphics areas: defining a good quality parameterization is still a tedious process, requiring experienced and skilled users.
In this paper, we specifically address the problem of finding automatically a parameterization of a reasonably small portion of the surface. This method is simple and efficient, which makes it suitable for interactive applications. It is also continuous in terms of the user input, so that it is stable during animation and interaction. Finally, it only requires the user to define a small set of intuitive parameters, mostly position, size and orientation of the area to be parameterized. Thus, it can be used by inexperienced users and easily connected to existing applications.
We show how our method can improve 3D-paint systems and decal placement on surfaces. The same approach can also be used to adapt 2D image processing tools to geometry filtering.

Keywords: Parameterization, Mapping, Polygons, 3D Paint, Interaction.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Viewing algorithms

1. Introduction

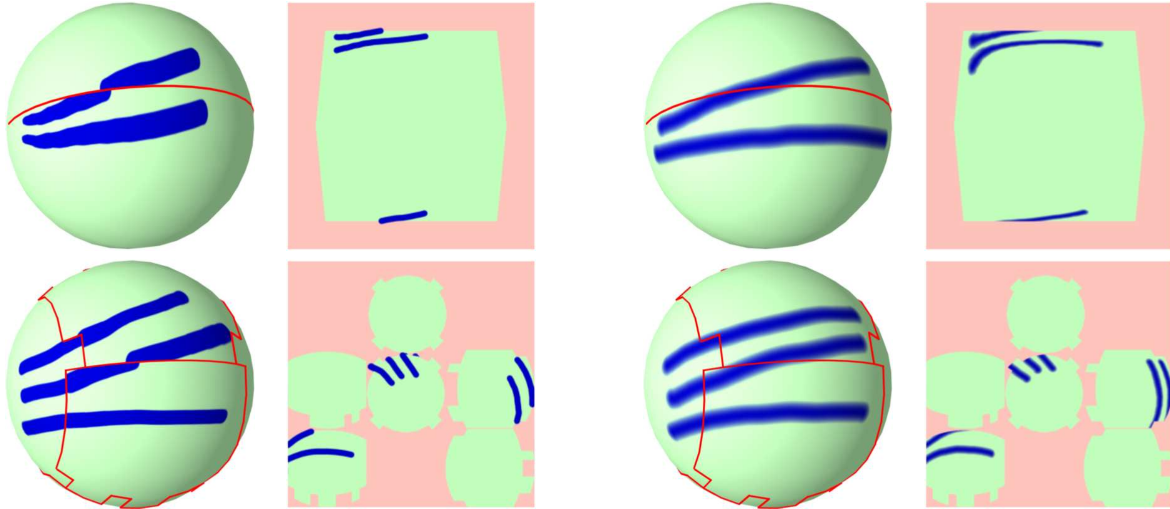
While a large number of commercial 2D paint tools are available [17], direct painting on surfaces is still a challenge. One reason is the fact that images are made of a collection of very organized samples: the pixels form a rectangular grid. Every pixel has a natural definition of neighbors, orientation, and size. All the paint operations and image transforms rely on this assumption.

On the contrary, 3D surface data is very unorganized. In general it is impossible to define a consistent global coordinate system on the surface. Traditionally, people have been using a UV parameterization of the whole surface, which always introduce distortion and discontinuities. A constrained optimization algorithm can be used in order to help the user [11, 12, 13, 14] define the parameterization, but the process is still tedious, and requires human intervention. Some work has been done to try to define the necessary texture seams [2, 9, 13], but so far, no satisfactory automatic method have been proposed.

Turk [21] and Witkins [25] proposed at the same time to gener-

† Alias|Wavefront, 210 King St. E. Toronto, ON M5A 1J7, Canada jmaillot@aw.sgi.com

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org



Painting directly in the texture image introduces two types of artifacts. The top row shows a sphere mapped with only one UV piece. Constant size strokes in texture space result in varying size on the 3d object. Furthermore, texture seams (shown here as red lines) clip the strokes sharply.
In the bottom row, the UV mapping deformation has been improved, using 6 plane projections. Stroke width preservation is enhanced, but the introduction of additional texture seams makes the overall result still not acceptable.
The images on the left represent the mapped 3D model, those on the right the texture that the user painted. The pink pixels represents the unused texture areas, while the green part is mapped on the 3d geometry.

Figure 1: *Artifacts when painting the texture.*

Using a local parameterization for each brush stamp, and accumulating stamps into the texture let the user paint on the geometry regardless of seems and UV mapping distortion. We painted strokes similar to the ones in Fig. 1 using our system.

Figure 2: *Using local adapted parameterization.*

ate textures directly on the surface. This idea has been reused and enhanced several years later [6, 7, 22, 23, 24]. This provides an elegant solution when the user want to cover a surface with a pattern. A program can automatically generate a texture on the whole surface, or a portion of it. Nevertheless, those methods are not adapted for 3D paint or decal placement, where the user wants to control precisely how the texture will look like on the object.

The generalization of paint operations to handle non-regular sampling and texture seams is a difficult process, especially when operations like smearing or blurring are involved. Most commercial packages [4, 8] get around the problem by requiring the user to define a *good* parameterization up-front, and by painting directly into the 2D texture. As shown in figure 1, this introduces two types of artifacts: a deformation of the paint stroke, as well as sharp clipping along texture seams. Adapting the parameterization to reduce the deformation necessitates more seams, and creates more artifacts of the second type.

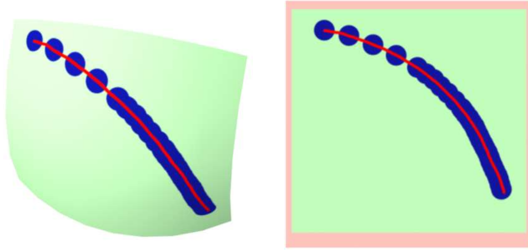
Instead of using a single global parameterization for the painting process, we propose to build on the fly many local ones, which are small enough to minimize distortion, avoid seams and be easily computed. Yet, they must be large enough to allow non trivial paint operations. It is also necessary that the parameterization vary smoothly as the user paints.

A global parameterization is still necessary to produce a single texture, in order to feed the result in a conventional graphics package, as shown in figure 2. But when using the appropriate algorithm to render the texture, like the one described in section 8, texture seams and mapping distortion are not an issue any more.

2. Overview

We will use 3D painting as the main example throughout this paper; section 10 shows additional applications of this technique.

The purpose of a paint program is to modify an image according to the user input. The user can draw curves called *strokes*, either directly on the texture or on the 3D surface. Each stroke is discretized into a series of samples, as shown in figure 3. In texture space, the sample is just the current UV position. When painting on a 3D surface, the sample is a 3D position, generally determined by intersecting the view ray under the mouse position with the 3D geometry. We call this 3D location the *hit point*. The paint program then performs a simple operation for each of such sample, which consists of painting a single *stamp*.



In paint systems, the user draws the red path, on the texture, or in the 3d view. The path is sampled, and for each sample, a stamp (blue disc) is painted into the texture. The accumulation of many stamps creates the impression of a solid thick line, the stroke. In the upper left corner, the sampling spacing has been artificially increased to distinguish each individual stamp.

Figure 3: *Paint strokes and stamps.*

We propose to introduce an intermediate coordinate space for each stamp, which is more appropriate than the texture space used in commercial packages. We compute a local parameterization, which is just large enough to cover the stamp, so that we can limit the texture mapping distortion, and avoid all texture seams. Our method will ensure that two successive stamps will have consistent parameterization in the overlap area, and that the computations are simple enough to cope with interactive feedback.

Let's call r the stamp radius, computed in world space. Our algorithm is made of the following steps:

1. Compute the hit point H, the three vertices V_1, V_2, V_3 forming the triangle H belongs to, and the corresponding barycentric coordinates α_i of point H.
2. For each vertex V_i:
   a. Find the size s_i of the largest connected edge.
   b. Compute the list $\mathcal{L}_i$ of all triangles, which are at a distance at most $r + s_i$ of V_i.
   c. Find the optimal parameterization $\mathcal{P}_i$ for $\mathcal{L}_i$, using an existing relaxation method.
   d. Compare the parameterization shape with the stamp outline. Possibly add or remove triangles, and repeat step 2.c. while the domain changes.
3. Rotate, translate and scale the parameterizations $\mathcal{P}_i$ into a common coordinate system.
4. Interpolate the parameterizations $\mathcal{P}_i$ using barycentric coordinates α_i for all common triangles in the three lists $\mathcal{L}_i$.
5. Paint the texture using the interpolated local parameterization and a global texture mapping function.

3. Determining the hit point.

For small objects, the application can explicitly test the intersection of the view ray with every triangle of the surface. The closest intersection to the camera is kept.

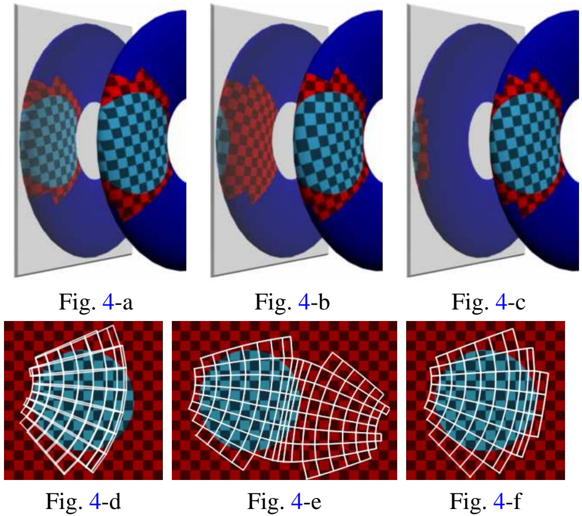


Figure 4: *When part of the surface silhouette projects inside the brush profile, both the front and back part of the object are included. Figure 4-a shows the brush projection on a flat torus; a mirror has been placed behind the torus to show the back part. Figure 4-b is the result of the optimization. In Figure 4-c, the large unnecessary part on the back has been culled, and the smaller neighborhood optimized again. Figures 4-d,e,f show the corresponding stamp spaces.*

For larger objects a brute force approach is too time consuming to achieve interactive rates. It is possible to use the previous hit point as a hint to accelerate the search: in most situations, successive hit points belong to the same, or neighboring triangles. The algorithm presented here looks for an intersecting triangle, using a breadth first search on the triangles, starting from the triangle of the previous hit.

This method is less accurate, because it does not guaranty to return the hit which is the closest to the eye, for concave objects. Another potential drawback of this method is that the computation time will vary, depending on how many triangles need to be traversed before a hit point is found.

Nevertheless, this algorithm is simple to implement, and does not require to precompute additional data structures like octrees or BSP's. It has been successfully used by artists in commercial packages like Maya [15], for direct interaction on surfaces.

4. Defining the neighborhoods $\mathcal{L}_i$

The second step in our process consists in defining a subset of the surface which be large enough to represent the stamp. As only the part inside that stamp area will be used, we must avoid selecting a large part of the surface. The reason is twofold: firstly, this would add more variables to the

optimization step and make the process slower. Secondly, this defines more constraints to the optimizer, and will result in more distortion in the interesting area. As an example, the smaller region mapped in figure 4-c results in a better mapping along the torus silhouette than in figure 4-b.

The main problem here is that the exact shape of the part of the model that will be covered is not known until the optimization is run. We propose one heuristic which proved to be simple to implement and efficient, yet yielded satisfactory results for users.

We compute a projection along the normal of the considered vertex. All the connected triangles which intersect the brush shape when projected into the brush plane are kept. Figure 4-a shows an example of such an area. As a projection will always reduce the size of triangles, the optimization process will tend to expand the UV domain, which generally will ensure that the whole stamp will be covered.

This does a good job on most surfaces, but may produce too large neighborhoods. In particular, when some part of the object silhouette is projected inside the brush, like in figure 4, both the front and back part of the object are selected. Section 6 explains how we resolve this issue.

5. Optimization technique

We need a fair parameterization, that preserves the lengths and angles as much as possible, to have the stamp warp along the geometry. This can be characterized as minimizing a distortion function [5, 14]. When the neighborhood is very small, a planar projection can be sufficient. But for useful stroke thickness, this may distort the stamp too much: we need to improve the mapping from the stamp to the surface. Several optimization methods have been proposed [11, 12, 14], but they are too slow for our interactive framework.

We implemented the cost function described in 2002 by Levy [13], and optimized it with a standard conjugate gradient method [19]. The fact that this function is linear makes it easy to implement and fast enough to optimize for interactive applications.

It is important to note that any other optimization algorithm with similar performance could be used in combination with the presented method.

6. Refining the neighborhoods

After the optimization, the resulting mapped area is generally too large, or sometimes misses a few triangles. The right side of figure 4-e shows an example of a large unnecessary mapped area.

In this step, all the triangles totally outside the stamp area are removed. In addition, the border of the mapped region (all the edges belonging to only one mapped triangle) is computed. For every border edge that intersects the interior of the stamp, we check the 3d geometry to determine if this is a geometry boundary also. When this is not the case, the missing triangle can be added. It must be noted that this case is very rare, and could be virtually eliminated by increasing slightly the brush size.

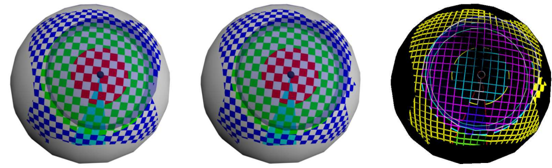


Fig. 5-a Fig. 5-b Fig. 5-c

Figure 5: *Two successive frames 5-a and 5-b from an animation where the stamp radius increases. All the triangles with valid parameterization are textured. The transparent sphere centered on the black dot represents the stamp size. Only one triangle was added on the right side, but this is sufficient to produce a noticeable popping effect during an animation. Using only discrete radius values, and interpolating the corresponding parameterizations eliminates this artifact. Fig. 5-c is an image difference between 5-a and 5-b.*

After the neighborhood has been adjusted, a new optimization is ran. In order to get the best result, this process can be repeated several times, until the neighborhood does not change any more. But this requires to take some precautions to avoid infinite loops where triangles removed in one step would be added again in the next. In most cases though, a single adjustment is sufficient. Our algorithm has been written to only remove triangle in the first adjustment, and repeatedly add triangles until the region does not change any more.

7. Ensuring continuity

We need to ensure that the local parameterization varies smoothly when the user moves the center of the the brush along the surface. For 3D paint, when two successive stamps overlap, the painted stroke should result in the proper accumulation of many stamps. Coherency is critical to ensure good quality results, especially when using textured brushes, or operations like smearing. Also, the user placing decals on a surface does not expect sudden changes as the decal is dragged.

In order to avoid discontinuities in the parameterization when the neighborhood is modified, we only compute a finite number local parameterizations, and interpolate them. We used a piecewise linear interpolation, where the parameterizations are computed for each vertex of the triangle mesh.

Each hit point keeps track of the three vertices forming the triangle it belongs to. The parameterization process defined above is ran for each of those three vertices, but the size of

the stamp is increased by the length of the longest edge connected to that vertex. The reason is that the parameterization must cover all the triangles required when the brush center moves anywhere in any connected triangle.

The three parameterizations are translated, rotated, and scaled to ensure the best match of the current triangle, using a least square minimization of the distances of the three vertices. As for 3D mesh morphing [1], fitting the orientations is critical to avoid UV shrinking during the interpolation.

The final parameterization is then interpolated using the barycentric coordinates of the hit point in the triangle. As each vertex parameterization does not depend on the hit triangle, we can guarantee that the result will vary smoothly, even when the brush crosses an edge between two triangles.

It must be noted that the domains of the three maps are not exactly the same. Only triangles which have a valid parameterization for the three vertices are interpolated. But as the brush size has been increased by the length of the longest connected edge, the common domain is large enough to fully cover the stamp.

For some applications the stamp size will remain constant. This is the case when painting constant width strokes, or dragging a decal at the surface of a model. But the stroke width can also be tied to the stylus pressure for example, and the user may experience sudden jumps in the parameterization. This can be seen in figure 5 where a small change in radius results in a noticeable checker pattern change.

The problem is that increasing the stamp size changes the neighborhood, which results in a different optimization solution. The way to avoid this problem is to compute the vertex parameterization only for discretized size values. The final result is the linear interpolation of lower and upper discretized size value. This almost doubles the computation time, as six local parameterization must be computed each time instead of three, so it is desirable that users can disable this feature when not necessary.

Applications can implement different schemes to define the size values. The difference between two successive discreet values must be small enough to avoid too large differences between the two neighborhoods, but not so small that the two neighborhoods will be identical. A constant increment, proportional to the average edge length in the object gives satisfactory results.

When the parameterization is computed, we translate it so that the hit point always match a specific location in UV space, generally the image center. Each vertex may also contain a desired direction for the U axis. This vector can be defined by the user, computed as a cross product between the normal and a fixed direction, or set as the stroke tangent for a 3D Paint application. We compute the three rotation angles necessary to align the U axis with each direction defined for the three vertices. The final rotation is computed using the barycentric interpolation of those three angles to ensure a smoothly varying orientation.

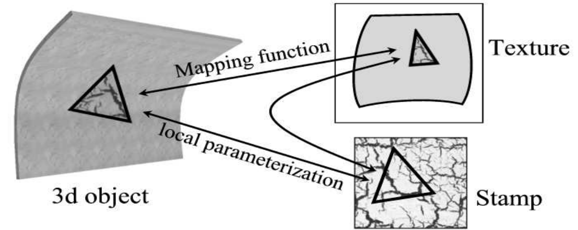


Figure 6: *We define an optimal mapping between the stamp space and the 3d model. The user already defined a global UV mapping for the object. Combining both mappings allows us to paint the texture properly, so that user sees a constant width stroke on the surface.*

Our solution is piecewise linear. It is C^0 along the edges of the triangles and for the scale values that change at least one neighborhood, and C^∞ in the rest of the input domain.

8. Rendering into the final texture

After the parameterization is defined, all the stamps must be accumulated into a single texture. Any global parameterization can be used, which may contain seams and large texture distortion.

Each vertex $V_i = \{x_i, y_i, z_i\}$, is associated with its global texture coordinates $U_i = \{u_i, v_i\}$, and its local parameterization $S_i = \{s_i, t_i\}$. For each triangle, the stamp texture is transfered from the stamp space into the texture space by combining the local and the global parameterization, as shown in figure 6. This process is equivalent to rendering a flat object, where the vertices are defined as $\{u_i, v_i, 0\}$ with the texture coordinates $\{s_i, t_i\}$. Conventional scanline rendering algorithms can be used.

Figure 7 shows that this process distorts the stamp texture in order to compensate for the global texture seams and distortion, so that the strokes width on the 3D geometry is preserved.

9. Coding optimizations

There are several implementation details which dramatically increase the efficiency of the calculations.

The first part of the algorithm determines the local neighborhood to consider. The hit point and the triangle it belongs to are known at this stage. In order to avoid searching through the whole object, triangles are traversed in a breadth first order, and culled as soon as they do not projects into the stamp. The method complexity is proportional to the number of triangles under the stamp, regardless of the object size.

Fig. 7-a

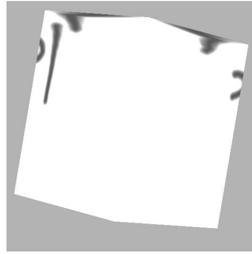


Fig. 7-b

Figure 7: *Rendering individual triangle in the texture, using the appropriate transformation between the stamp and the texture eliminates stretching and seams artifacts. It would be very difficult for a user to paint directly in the texture to achieve the same result.*

Determining the local neighborhood involves one optimization after each modification of the triangle list $\mathcal{L}_i$. The first optimizations are ran with very few conjugate gradient iterations to quickly produce a coarse result. When $\mathcal{L}_i$ is determined, a smaller stopping threshold is used to finally produce the accurate result.

We cache the last three vertex parameterizations, or the last six when the stamp size may vary. In practice, successive hit points belong to the same triangle, and when this is not the case, they often belong to adjacent triangles. In this case the two (or four) vertices forming the common edge can be reused. This allows to only run the interpolation step most of the time, and limit the number of optimization calls to the strict minimum.

As an example, the face model of figure 8 contains 6500 faces. Recomputing the parameterization at the size of the figure can be done in real time on single processor Pentium II machine.

10. Applications

Figure 9 shows an example of 3D painting. Each stamp used our local parameterization to smoothly follow the surface. In figure 10 we used a blur and smear tool. Both operations require a local flat image to work. The color information is extracted from the model and rendered in a temporary stamp image, using the inverse of the rendering function described in figure 6. A standard image processing filter can be applied, and the stamp is painted back onto the model.

Figure 11 shows an implementation of a wrap deformer tool. The cube is moved and bent along the face surface using the local parameterization. In this example, we mapped the X and Y space coordinates of the cube to the local U and V, and moved the vertices by their Z values along the local normal.

Fig. 8-a

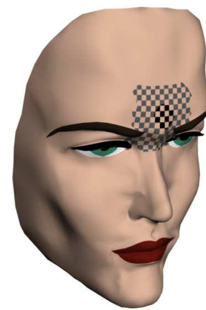


Fig. 8-b

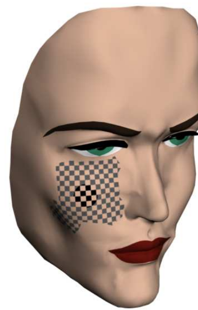


Fig. 8-c

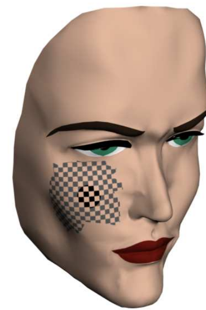


Fig. 8-d

Figure 8: *Extract from a test animation. The hit point is animated and moves along the face surface. Both rows show successive frames when the valid domain changes. The differences between the parameterizations are very subtle, and appear smooth when playing the movie.*

11. Limitations and future work.

The major problem we encountered is when the stamp covers an area of the model which cannot be unwrapped easily. One reason may be that the object contains a lot of very curved areas. This is happening for example just above the upper lip of the model in figure 8. The base of the nose is very intricate and difficult to map. Our method produces some local parameterization, but it is very distorted. This is a limitation of the model shape, and not much can be done. In practice, users will generally not run into this problem, because it means that the brush size is too large to produce accurate results. It might be useful, though, to detect this case, and either warn the user or automatically resize down the brush temporarily. More test should be done in this case to determine the ideal workflow.

It is also possible that the brush becomes so large that the considered neighborhood completely wraps around the object. In figure 12-a, the neighborhood of the hit point is made of a single set of triangles forming a connected band around the cone. This shape is very hard to optimize as a

Figure 9: *Face painted with our brush. The stroke follows nicely the surface.*

Figure 10: *The top stroke was faded using a blur tool. The cheek one modified with a smear operation.*

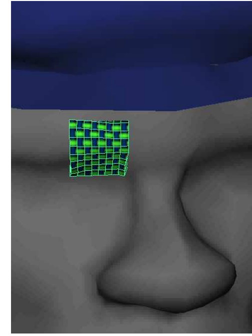


Fig. 11-a

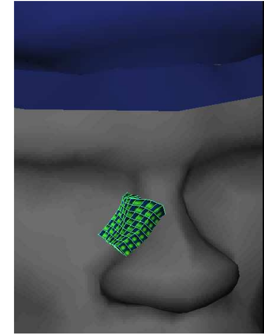


Fig. 11-b

Figure 11: *Extracts from a warping deformation test. A cube is moved and deformed along the face.*

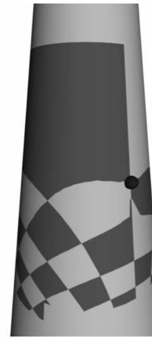


Fig. 12-a

Fig. 12-b

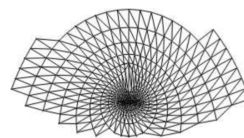


Fig. 12-c

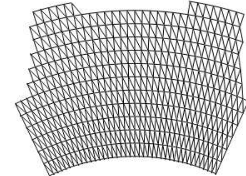


Fig. 12-d

Figure 12: *When the neighborhood is not simply connected, our algorithm may behave badly. Adding a seam at the back on the cone solves the problem, as can be seen in the right column.*

whole, which leads to the high amount of distortion. The parameterization shown in figure 12-c is very bad. In simple examples like this one, it is possible to insert a seam for all edges which represent a maxima of the distance to the hit point along the surface. This gives the solution shown in the right column. In general, it is desirable to ensure that the neighborhood is always simply connected.

We only implemented a piecewise linear interpolation of the UV values. When the brush size varies, it would be easy to compute the parameterizations for 4 size values instead of 2, and use a cubic interpolation, which would be smoother. Similarly, we could compute and cache the parameterizations for more vertex positions and use a higher order interpolation for the brush center. So far, we have not found a good interpolation scheme for vertices, but some subdivision surface methods could apply to this solve problem.

Finally, we would like to exploit this technique to filter or resample the geometry. We believe that working in a local flat space can allow simpler implementation of geometry smoothing, and better quality resampling. This technique could also improve geometric tools like the shape cut and paste presented by Biermann [3].

Acknowledgments

Many thanks to A|W for the support of this research project. In particular, the feedback of the developement teams, internal testers and expert users has been very valuable to validate the results and make sure they were practical. The face models used for testing were built by Corinne Chaix.

References

1. **M. Alexa, W. Müller.** *Representing animations by principal components.* Eurographics 2000 Proceedings, 19(3):411–409, 2000. 5
2. **C. Bennis, J-M. Vezien, G. Iglesias, A. Gagalowicz.** *Piecewise surface flattening for non-distorted texture mapping.* Proceedings of SIGGRAPH 1991, pp 237–246, 1991. 1
3. **H. Biermann, I. Martin, F. Bernardini, D. Zorin.** *Cut-and-paste editing of multiresolution surfaces.* Proceedings of SIGGRAPH 2002, pp 312–321, 2002. 8
4. **Deep Paint 3D.** *Right Hemisphere Ltd.*. www.righthemisphere.com, . 2
5. **M.P. Do Carmo.** *Differential Geometry of curves and surfaces.* Prentice-Hall, Inc, Chapters 2-5 and 4-2, 1976. 4
6. **A. Efros, T. Leung.** *Texture syntesis by non-parametric sampling.* Proc. ICCV, 1039–1046, 1999. 2
7. **A. Efros, W. Freeman.** *Image quilting for texture synthesis and transfer.* Proceedings of SIGGRAPH 2001, pp 341–346, 2001. 2
8. **Flesh.** *D 'n A software* . www.dnasoft.com/products/flesh/, . 2
9. **J. Hart, N. Carr, J. Maillot.** *The Solid Map: Methods for Generating a 2-D Texture Map for Solid Texturing.* Manuscript. http://graphics.cs.uiuc.edu/ jch/papers/pst.pdf, January, 1999. 1
10. **A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, D. Salesin.** *Image analogies.* Proceedings of SIGGRAPH 2001, pp 327–340, 2001.
11. **B. Lévy, J.L. Mallet.** *Non distorted texture mapping for sheared triangular meshes.* Proceedings of SIGGRAPH 1998, pp 343–352, 1998. 1, 4
12. **B. Lévy.** *Constrained texture mapping for polygonal meshes.* Proceedings of SIGGRAPH 2001, pp 417–424, 2001. 1, 4
13. **B. Lévy, S. Petitjean, N. Ray, J. Maillot.** *Least squares conformal maps for automatic texture atlas generation.* Proceedings of SIGGRAPH 2002, pp 362–371, 2002. 1, 4
14. **J. Maillot, H. Yahia, A. Verroust.** *Interactive texture mapping.* Proceedings of SIGGRAPH 1993, pp 27–34, 1993. 1, 4
15. **Maya.** *Alias|Wavefront*. www.aliaswavefront.com, . 3
16. **H.K. Pedersen.** *Decorating implicit surfaces.* Proceedings of SIGGRAPH 1995, pp 291–300, 1995.
17. **Photoshop.** *Adobe* . www.adobe.com, . 1
18. **E. Praun, A. Finkelstein, H. Hoppe.** *Lapped texture.* Proceedings of SIGGRAPH 2000, pp 465–470., 2000.
19. **W. Press, S. Teukolsky, W. Vetterling, B. Flannery.** *Numerical Recipes in C.* Cambridge University Press, pp 420–425, 1992. 4
20. **G. Taubin.** *A signal processing approach to fair surface design.* Proceedings of SIGGRAPH 1995, pp 351–358, 1995.
21. **G. Turk.** *Generating texture on arbitrary surfaces using reaction-diffusion.* Proceedings of SIGGRAPH 1991, pp 289-298, 1991. 1
22. **G. Turk.** *Texture synthesis on surfaces.* Proceedings of SIGGRAPH 2001, pp 347–354, 2001. 2
23. **L. Wei, M. Levoy.** *Fast texture synthesis using tree-structured vector quantization.* Proceedings of SIGGRAPH 2000, pp 479–488, 2000. 2
24. **L. Wei, M. Levoy.** *Texture synthesis over arbitrary manifold surfaces.* Proceedings of SIGGRAPH 2001, pp 355–360, 2001. 2
25. **A. Witkin, M. Kass.** *Reaction-diffusion textures.* Proceedings of SIGGRAPH 1991, pp 299-308, 1991. 1