Painting on Moving Charts

Shadi Ashnai and John C. Hart

Department of Computer Science, University of Illinois at Urbana-Champaign
{sashnai2, jch}@uiuc.edu

Abstract
In this paper we propose a system for painting on surfaces. This system is based on moving charts. At each moment, a local patch is selected on the surface. The patch is formed such that it covers an area on which an artist can easily paint. This patch can be subjected to move in either direction according to the artist's request. For each patch, an optimized 2-D chart is generated and displayed to the user. The user can then paint on this chart. Texture, which is stored in a global atlas is updated due to the user modification. For our application, charts must have as little distortion as possible and chart boundaries should be as smooth and convex as possible. Therefore, we chartify local patches using a low-distortion parameterization while preserving the boundary convexity and smoothness.

1. Introduction
Texture mapping has long served the computer graphics community as a convenient shortcut for creating the appearance of detail on coarse geometric models [BN76]. It relies on a mapping from the surface to a 2-D texture commonly implemented in real-time rasterized meshes through the assignment and interpolation of per-vertex texture coordinates. A variety of approaches have automated the generation of texture coordinates, e.g. [SCOGL02, GGH02], typically minimizing induced distortion to provide an isometric mapping that aims to preserve the proportions and angles of the texture image on the surface. In an effort to further reduce distortion, this mapping is commonly decomposed into a texture atlas of multiple charts.

A texture is often created for a particular geometric model. To this end, several tools facilitate the ability to paint directly on an object, rendering paint strokes into the texture using the texture coordinates under the surface positions of the stroke [Hae90, IC01, CH04].

Even though sophisticated surface painting tools are available, artists tend to prefer painting on the texture image over surface painting when texturing a production model for the following three reasons. First, surface painting requires numerous object rotations and is particularly difficult in concave regions. Second, the flat domain of a texture image better captures the subtle inflections of an artist's brush strokes than does the projection of a surface undulating in 3-D. Third, many powerful tools exist for 2-D drawing, painting and design that outnumber and outweigh those available for 3-D surface painting.

Existing tools for texture mapping rely on cutting seams to reduce distortion and often result in a large number of small charts packed efficiently into scattered positions of the texture atlas that further complicate the mental correlation between the texture and the surface. Maintaining this correlation adds time to the painting process and can inhibit expressive painting.

We propose a new, more task-conformant approach for texture painting that allows the artist to drag a low-distortion chart around the 3-D surface and paint texture onto a flattened 2-D version of the chart. By flattening only a local chart we avoid distortion more easily than in a global parameterization. The chart can be interactively dragged and resized across the surface to provide the artists with a canvas that most appropriately supports the current painting area of focus.

2. Previous Works
Many approaches have been introduced during the past years for decorating surfaces. Some directly texture the 3-D surface. Solid texturing 3-D surfaces, for example, is one of the most known algorithms in this category [Pea85, CH02]. Similarly, Debry et. al [DGPR02] have introduced a system that stores and modifies the texture in a 3D Octree.
all developed 3-D systems that provide users 3-D brushes to paint directly on the 3-D model. More recently, [CH04] supported direct surface painting using a dynamically reconfigurable 2-D texture atlas to manage detail. However, it is not always convenient for artists to draw in a 3-D space. Therefore, most of the texture mapping techniques are based on parameterizing the surface either locally or globally.

Texture mapping algorithms are available for texturing the whole surface of an object using a small texture sample. [WL01] have used a neighborhood search algorithm according to the flattened local neighborhood of the object. They use the closest match to grow the textured region. [ZG03] later introduced Jump Maps to accelerate the process of similar region searches.

Among 2D texture mapping methods, most use a global parameterization of the surface, while others preferred a local parameterization that focuses on a region of interest. For example, [LM94] is based on corresponding the global parameterization of the surface with the available texture and then mapping the texture on top of the surface using the user-defined correspondences. On the other hand, some algorithms use local patches of the model, commonly homeomorphic to a square. [Ped95, Ped96] have introduced Pachinos that construct a surface parameterization of the 3-D surface. In a similar vein, [SGW06] have introduced Discrete Exponential maps.

Another approach is to take several images and use an algorithm that finds an appropriate mapping between the images and the surface of the object. MatchMaker by [KSG03], and TextureMontage by [ZWT*05] are two recent works in this category. They both define several corresponding feature points and map the texture regions to the surface.

3. Moving Charts

Our system is based on local moving charts. Each chart is a 2-D parameterization of a local region of interest that is used by the artist for mapping texture on the surface. In this section we describe how we choose the region of interest, how this region is parameterized to the plane, and how we update the local chart according to user manipulation.

3.1. Creating the Local Chart

Local moving charts are formed by growing a region of interest around the seed triangle and parameterizing the new triangles simultaneously. Our local parameterization algorithm is based on a greedy iterative low-distortion method introduced by [SCOGL02, CHCH06]. At each iteration, patch front is the list of non-flat triangles adjacent to the boundary and a free vertex is a non-flat vertex of any of these triangles. Grade or rank is a scalar metric assigned to each free vertex and is used to determine the order of the vertex additions.

Chart creation starts with the seed triangle as the only flattened triangle. At each iteration more triangles are added to the chart by flattening the best free vertex. The algorithm consists of the following steps:

- Choose a seed triangle
- Map the seed triangle to the 2-D plane
- Iterate until new vertices can be added to the chart
  - Update the chart boundary and list of free vertices
  - Assign a grade to each free vertex
  - Choose the free vertex with lowest rank and add it to the chart if it meets all of the parameterization criteria

In the following section we describe the metrics we use to compute a final rank on each free vertex.

3.1.1. Vertex Metrics

The parameterization algorithm introduced in [SCOGL02] is used to pre-compute the 2-D location of the free vertex. The average distortion incurred on triangles adjacent to the vertex is then computed. A rank is assigned to the vertex such that the vertex with lowest rank causes the least distortion to the chart.

In the chart creation algorithm described in [CHCH06], additional metrics are used to define the rank of free vertices. Boundary smoothness and geodesic distance to the seed triangle are two major metrics we adopt from this work.

In addition to these metrics we compute the perimeter increase incurred by flattening each vertex. We also take into account the patch front degree of each vertex when computing vertex rank. Below we describe in more detail how we compute each of these metrics for a free vertex and how we compute the final rank.

Figure 1 shows results of our local parameterization. In this figure (as well as all other results shown in this paper), low distortion triangles are blue, becoming green and then red with increasing distortion.

**Distortion Metric:** Triangle distortion is determined using the singular values of the Jacobian of the affine transformation between the original and the parameterized triangle. Smallest and largest of the singular values, $\gamma_{\min}$ and $\gamma_{\max}$, are the amount of shrinking and stretching incurred to the triangle. The distortion is then evaluated as the maximum of $1/\gamma_{\min}$ and $\gamma_{\max}$. As in Sorkine’s algorithm, we define the distortion of a free vertex to be the maximum distortion of its adjacent triangles.

**Geodesic Distance Metric:** We define this metric to be the length of the shortest path of edges between the vertex and one of the three vertices of the seed triangle. Giving higher priority to the vertices closer to the seed triangle, will force the region grow in a more circular pattern. This will help us construct a parameterization with a less jagged boundary.
Normal Deviation Metric: We approximate a normal to the boundary in the plane at every free vertex. We define the normal deviation metric to be \(\sin\) of the angle between this approximated normal and the ideal normal for the chart at the location of the vertex. Assuming we would like to produce a round or square chart, we can easily compute the direction of the ideal normal at each point.

Perimeter Increase Metric: The perimeter increase is simply the difference in total perimeter length before and after a vertex is added. Note that this may actually be a decrease. Adding vertices that reduce this factor helps reduce cracks and slivers in the boundary.

Patch Front Degree: In addition to the above metrics, practice has shown that adding vertices that have higher degree of adjacency with the boundary earlier will result in less overall distortion to the final chart. If flattening a free vertex adds more than one triangle to the chart we reduce its rank so that it is added to the chart in an earlier stage.

Once all of the above metrics are computed for a free vertex, we combine their values to assign a final rank to the vertex and define an order for growing the chart. We use a linear combination of these metrics to define the rank. Weights for this combination are initially uniform and can be modified by the user at any time.

Each iteration of the algorithm, adds the vertex with minimum rank to the chart. Based on the definition of our rank, such a vertex does not cause a large amount of distortion to the chart. It is also not too far from the seed triangle. Moreover, it does not increase the perimeter of the chart by a large amount, it might even decrease the perimeter length by smoothing out the jagged boundary.

We compare the results of our parameterization to one that uses only the distortion metric. Parameterization of the side of the cow (above) and the back of its head (below) are shown in the figure. Using the combined metric we get a chart with smoother boundary with little additional distortion to the chart, overall.

3.2. Updating the Local Chart

The chart is updated whenever the user moves it around the surface to paint on different regions or wants to change the size of the chart she is working on. We introduce incremental updates in order to increase the speed of user interaction with the system.

3.2.1. Moving the Local Chart

When the chart is moved, the seed triangle and hence the local region needs to be updated. The new region could be grown around the new seed triangle from scratch. However, since movements are often such that the new seed is close to the old seed, we may have a significant number vertices shared by the new and previous patches. Regrowing the patch for each movement of the seed triangle would cause long delay in our interactive system. Instead, with respect to each move we detect vertices that should be excluded from the patch and also grow the patch by adding any vertex within the viewing window that meets our growth criteria.

Deletion: To detect vertices that need to be deleted from the patch, we check boundary edges to see if any of them are outside the view window. If so, starting from that edge we determine the face beside that edge that is marked as flat yet has to be removed from the flat patch. We mark this face as non-flat and check its edges to update the boundary list. Using a simple DFS algorithm on faces, starting from this face, we can remove one set of out-of-view faces from the chart.

Addition: Adding new vertices is done in exactly the same way as expanding the initial chart. We continue adding new vertices and updating ranks until no more vertices can be added. To update boundary edges, we only need to check the local area around the newly added vertex. We do not worry about the boundary at these iterations. We can simply traverse the flat region once at the end and update the list of edges in the boundary.

3.2.2. Modifying Size of the Local Chart

Our system lets the user modify the size of the local chart. As examples in figure 2 illustrate, on highly curved regions like the tip of the finger, smaller charts have less seams and less distortion. As we enlarge the chart and get further from the seed triangle, not only higher distorted triangles are added to the patch but also the boundary becomes more jagged. On the other hand, on some elliptic regions, like the back of the hand, large size neighborhoods can be flattened without having a very high overall distortion or jagged boundary. Therefore, we can modify the chart size in order to shrink or stretch the region viewed to the user for painting.
3.3. Comparison with Conformal Parameterizations
We have compared our greedy parameterization results with conformal parameterizations of the same region using one of the intrinsic parameterization methods described in [DMA02]. Figure 3 shows the comparison of the conformal parameterization versus parameterization using our greedy algorithm on three different regions on the hand model. Two left columns show the conformal parameterization and the two right columns show the greedy parameterization of the same local region.

4. Painting on Moving Charts
The main goal of this system is to allow artists create or modify the texture of the mesh surface. Our system does not create a global texture atlas for the object and uses the local chart that is provided with the model. There exist tools that allow artists paint on the surface in 3-D environment. However, for some specific patches, it is not convenient to paint in 3-D.

Figure 4 illustrates an example of a hyperbolic 3-D patch and its parameterization. On the left, the 3-D patch on the object is highlighted, painting a shape on this region that is folded on two sides of the object, in a 3-D space, needs several movements of the object. Center image shows a conformal parameterization of the patch. As shown, the distortion does not follow a special pattern. Painting on such a parameterization will distort the texture in an unpredictable way. Chart shown on right has low distortion and is focused on the seed triangle. Therefore drawing patterns or pasting images on this chart is easier and also preserves the texture.

We keep track of a local texture map in the system, which represents the current local chart and the texture mapped into it from the global texture map. The global texture map might be automatically generated, for example as in [CH02, LPRM02], or manually by an artist. Using our system, artist paints on the local chart. Changes are later mapped to the global texture and applied on the 3-D object. Figure 5 illustrates an example of mapping a sprite on an object. Selected sprite contains a text that needs to be mapped to the 3-D model with as less distortion as possible.
In comparison to our parameterization technique, figure 6 shows a conformal parameterization of the exact same region. Same sprite as above is mapped on the chart created using conformal parameterization. We can see the distortion of the text on this surface compared to our algorithm.

![Figure 6: Result of modifying the texture of the local patch created by conformal parameterization. The selected patch(left), triangle distortions(center) and the final result for texture mapping(left) is shown in this figure.](image)

5. Conclusion

We have introduced a new system to support painting texture on a surface by dragging a local chart that can be flattened and painted in any concurrent 2-D paint application. We allow the artist flexibility to trade distortion for boundary simplicity depending on which serves their task.

Our mixture of these chart construction constraints occurs along the frontier of a greedy chart growth algorithm originally designed to minimize only distortion. Other methods likely exist for further chart optimization based on these constraint mixtures.

This paper describes the completion of a prototype technology behind a moving charts texture painting system. Our design was based on user evaluation of existing systems, but a user studies on this system would further validate the design choices of this system and perhaps provide additional directions for further research.

References


