Texture Mapping of Images with Arbitrary Contours

Nicolas Cherin Frederic Cordier Mahmoud Melkemi
LMIA, Université de Haute Alsace (LMIA, EA 3993)
4, rue des Frères Lumière
68093, Mulhouse, France
nicolas.cherin@uha.fr frederic.cordier@uha.fr mahmoud.melkemi@uha.fr

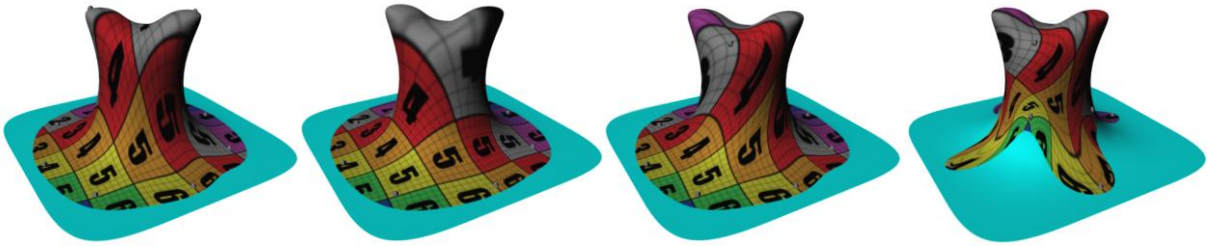

Figure 1. Local texture mapping with different sets of feature points.

ABSTRACT

Decaling is an intuitive paradigm for texture mapping in an analogy of attaching stickers on an object in the real world. This paradigm enables an artist to put decals directly on a 3D model after interactive manipulations such as modifying their positions, scales and orientations. In this paper, we present a novel method for multiple-constrained decaling. Given a region inside a texture together with a set of feature points in the region and a 3D model, our problem is to map the texture region onto the surface of the model in an intuitive manner, while satisfying the constrained imposed by a user-specified correspondence between a set of feature points in the region and the surface. We propose a solution for this problem. Our approach iteratively determines a portion of the mesh representing the surface while accordingly refining its parameterization, guided by the feature point correspondence.

Keywords

Texture Mapping, Parameterization, Polygonal Modeling.

1. INTRODUCTION

Texture mapping is a well-known technique for mapping an image onto the surface of a 3D model to enhance its visual appearance. This technique has been adopted for a broad range of applications such as special effects for the film industry that requires highly realistic models as well as the game industry for efficiently creating 3D models and virtual characters. The essential step of texture mapping is the surface parameterization of a 3D model, i.e. finding a one-to-one correspondence between the entire surface of the model and a texture.

A 3D model can also be decorating with several textures, that is, different images of arbitrary shapes are placed on the 3D model, each image covering a portion of the surface by locally parameterizing the region on the 3D surface that corresponds to each image. The metaphor can regarded as affixing stickers or decals to the surface of the model. This technique shows the possibility of texturing models by compositing images directly on the 3D surface, which is analogous to 2D image compositing that creates a new image by combining images from different sources by alpha blending. The texture mapping with local parameterization usually produces higher quality results than texture mapping with global parameterization, since local parameterization for a smaller number of triangles results in lower distortion compared to global parameterization for the entire surface.

The latest work related to local parameterization uses a discrete approximation to the exponential map [Sch06a] that parameterizes a circular region around a center point provided by the artist. As pointed by the authors, a disadvantage of their technique is that

the distortion of mapping increases significantly as the textured region is becoming larger, especially on surfaces with high frequency features. In addition, their technique offers limited control, that is, only the position of the center point, the scaling and the orientation of local parameterization can be specified by the artist. This method cannot be used for constrained texture mapping where we need to define a correspondence between multiple feature on the image and the 3D surface.

We present a method which is a generalization of the work of Schmidt et al., that is, the mapping of an image of an arbitrary shape onto the 3D surface given multiple corresponding pairs of feature points on the image and the surface. The input of our approach is a region of the 2D image that is bounded by a simple closed curve, and a set of feature points in the region and their counterparts on the 3D model. Our method computes automatically the region for texturing on the 3D model and its parameterization in an intuitive manner. Compared to the work of Schmidt et al., our approach offers two important advantages. First, the artist can use as many feature points as needed; our method ensures the exact matching of the features between the image and the 3D model guided by the feature point correspondence. Second, our method does not impose any limitations on the size and shape of the textured region on the model surface. Our method provides valid parameterization even when the textured region is very large and the surface of the model contains sharp features. For efficient texture mapping, we introduce a novel two-step parameterization that supports multiple feature correspondence and automatic computation of the 3D region for texturing. We show how to use this method to texture-map parts of the surface of a 3D model.

2. RELATED WORK

A variety of techniques have been proposed to help artists to decorate 3D models. We give a brief description of these techniques.

Surface Painting: Surface painting is one of the most common techniques for decorating 3D models; the artist creates a texture from scratch by drawing directly on a 3D model using painting tools such as a brush or an eraser. This technique has been well studied [Car04a] and many commercial tools for 3D painting are available [May05a]. However, surface painting is tedious and requires artistic skills to create a complete texture.

Texture Tiling: Another technique for creating textures on 3D model is to cover its surface with partially overlapping images [Pra00a] [Tur01a] [Wei01a] [Sol02a]. This technique is useful for completely covering a surface with repetitive applications of a pattern image. However, the use of this method is limited since it can only be applied for texture tiling.

Global Planar Parameterization: A large body of work on texture mapping has been devoted to global parameterization of surfaces, i.e. finding a bijective function between the entire surface of a model and a planar texture space. If the surface is topologically equivalent to a disk, then a planar parameterization is computed through an optimization that finds the position of vertices in the texture space such that distortion of the triangles is minimized [San01a], [Lev02a], [Des02a], [Flo03a], [Flo03b], [Mey02a].

If the surface of the model is not topologically equivalent to a disk, the surface is segmented into a set of disjoint charts, each of which is homeomorphic to a disc and parameterized independently of each other [Lev02a], [Zho04a], [Zha05a]. These parameterized charts are then packed into the texture space to collectively form a texture atlas.

The surface parameterization technique has been further extended to incorporate a feature correspondence between points in the texture space and vertices on the surface of the model. The feature correspondence is integrated in parameterization either as soft constraints [Lev01b] [Des02a] or hard constraints [Kra03a]. [Zho05b] further extends the constrained parameterization to allow the artist to generate a texture atlas from multiple images.

Since global parameterization is targeted for mapping the entire surface, distortion due to parameterization usually increases with greater surface complexity. Our approach is based on local parameterization, which aims at lowering the distortion by restrictively parameterizing the portion of the surface that is actually textured.

Local Parameterization: Unlike global parameterization, local parameterization is computed only for the region of the 3D surface that is to be textured. This technique is known as decal mapping, in reference to the metaphor of a decal (or sticker) affixed to the surface of an object. [Lev05a] have proposed a system supporting the interactive manipulation and composition of decals. The parameterization of decals is obtained with a planar projection of the 3D surface to be textured. While this method is simple and shows good computational efficiency, the planar projection of highly-curved surfaces results in significant distortion. [Sch06a] have computed the parameterization with discrete exponential maps, which significantly improves the quality of the parameterization compared to the planar projection. Still, the quality of the parameterization with this method is sensitive to high frequency features of the 3D surface to be

parameterized. Our system combines conformal mapping and 2D warping to robustly handle the parameterization of surfaces with high frequency features. Besides, our method allows users to introduce multiple feature constraints, which facilitates precise alignment between texture and surface features.

Recently, some researchers [Sun13a] have proposed an interactive interface for texturing 3D surfaces. With this system, the user specifies a local parameterization with a free-form curve drawn on the surface. Compared to their method, our approach offers higher level of user interaction. In their system, the texture image should have the shape of a strip and its mapping is achieved through the manipulation of a surface curve. In our system, the texture image can be of any shape and the mapping is controllable with an arbitrary set of feature points.

3. OVERVIEW

We provide a novel texturing technique that is powerful, yet easy-to-use for decorating 3D models with one or more textures. Basic operations for mapping a texture on a 3D model are cutting an image with a simple closed curve and pasting the result onto the surface of 3D model guided by a set of corresponding pairs of feature points, each constraining a vertex of the 3D model to a position in the image.

For the remainder of the paper, we refer by the texture space to the 2D space where the image and the simple closed curve are located. The object space is the 3D space containing the triangular mesh of the model (Figure 2(b)). An image region is a part of the image that is surrounded by the closed curve (Figure 2(a)) and a patch is a subset of triangles of the triangular mesh, each triangle corresponding to a triangle in the 3D model to be textured. The position of the patch vertices in the texture space is computed through the parameterization of the patch.

Our texturing method is comprised of three steps. We first find a set of connected triangles on the 3D surface that contains all the feature points and that region containing the triangles is homeomorphic to a disc. We place these triangles inside the closed curve in the image (Figure 2(c)). The patch is then grown iteratively by adding a number of triangles at a time. At each iteration, we reparameterize the modified patch to minimize the texture distortion while satisfying the feature point constraints. This process is repeated until the patch completely covers the image region bounded by the closed curve (Figure 2(d)). Finally, we transfer the image region onto the 3D surface exploiting the patch and the parameterization (Figure 2(e)).

These three steps are described in details in the Sections 4, 5 and 6 respectively. Several examples of models textured with our tool are shown in Section 7. We discuss about the limitations and the future work in Section 8.

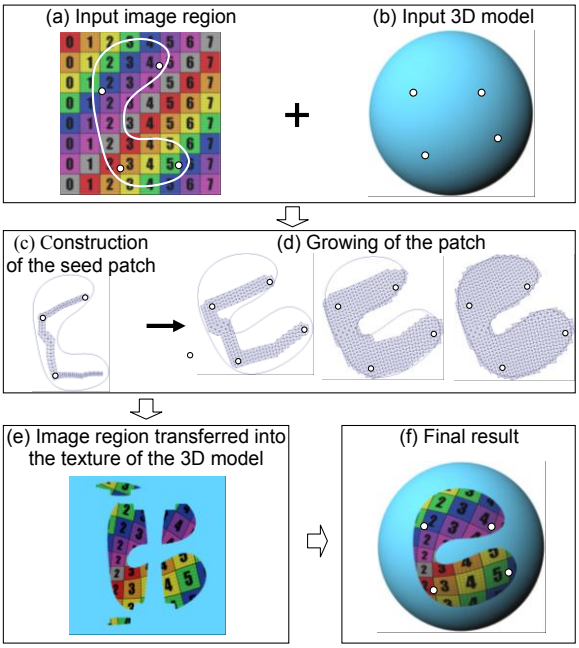

Figure 2. Overview of the texturing method.

4. BUILDING A SEED PATCH

The objective is to create a seed patch satisfying the feature constraints to bootstrap the patch growing. Specifically, the seed patch must be composed of a set of connected triangles containing the feature points; the position of feature points should be located at the given position and the vertices of each triangle in the patch should be inside the image region. In addition, since the image region is bounded by a simple closed curve, the patch should be homeomorphic to a disc.

In order to find the triangles on the 3D surface to build the patch, we first construct a 2D planar *feature-point graph* $\mathcal{G}_I$ in the image region; this graph has a set of vertices corresponding to feature points and a set of edges that are straight-lines joining a pair of feature points (see Figure 3(b)). Note that edges intersecting the boundary of the image region are not included in $\mathcal{G}_I$ (Figure 3(c)); the outline of $\mathcal{G}_I$ provides a rough approximation of the shape of the image region. We construct another graph $\mathcal{G}_S$ with the same connectivity as $\mathcal{G}_I$, on the 3D surface to obtain a rough approximation of the location of the patch on the 3D surface. Each edge of the 3D graph $\mathcal{G}_S$ corresponds to an edge of the 2D graph $\mathcal{G}_I$. Unlike a 2D edge, 3D edge represents the shortest Euclidean path in the triangular mesh that connects a pair of 3D feature points. Finally we use such path to find the triangles to construct the 3D patch.

4.1. Construction of the planar feature points graph in the image region

We first compute a triangulation of the feature points in the image region, employing a method which is essentially the same as the incremental Delaunay triangulation except that each edge of the triangulation lies completely in the image region. In order to identify the edges, we initially find all line segments, each connecting a pair of feature points and which do not intersect the boundary of the image region. These line segments are put in a priority queue ordered by their length. The line segments are then chosen one by one in sequence, starting from the shortest one, such that the line segments already chosen do not intersect each other. After adding an edge, we use edge-flipping algorithm [Hur99a], to flip edges which violate the local Delaunay criterion and do not intersect the boundary of the image region after the flip.

The resulting triangulation may be composed of several disconnected components due to the lack of line segments satisfying the non-intersecting requirement with the boundary. In this case, the artist is required to place additional feature points to obtain the triangulation.

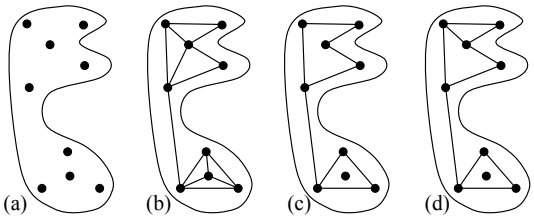

Figure 3. Construction of the feature point graph G_I: image region with the feature points (a), a triangulation of the feature points (b), removal of internal edges (c), decomposition into convex faces (d).

Next, we build a feature-point graph $\mathcal{G}_I$ by deleting all internal edges from the above triangulation (Figure 3(c)). Note that this may create feature points with no edges incident to it; these isolated feature points are always located inside a face. In order to simplify the computation of the initial parameterization of the seed patch (see section 4.3), we decompose every concave faces of $\mathcal{G}_I$ (regions bounded by edges) into convex ones by inserting additional edges (Figure 3(d)).

4.2. Selecting triangles to construct the seed patch

In this step, we identify the part of the surface of the 3D model to be textured. We first construct the graph $\mathcal{G}_S$ by embedding the edges of $\mathcal{G}_I$ onto the 3D surface. Given the 2D feature points $F_{I,j}$ and $F_{I,k}$ and their corresponding 3D feature points $F_{S,j}$ and $F_{S,k}$, the embedding of the edge $(F_{I,j},F_{I,k})$ is obtained by finding the shortest path on the mesh connecting $F_{S,j}$ to $F_{S,k}$.

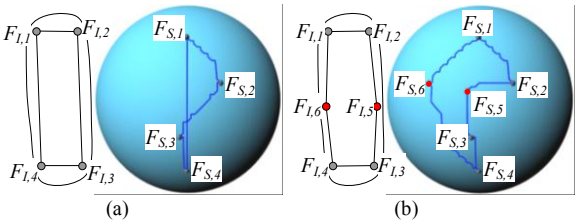

Figure 4. The paths of the face ($F_{S,1}$,$F_{S,2}$,$F_{S,3}$,$F_{S,4}$) intersect each other (a); two feature points $F_{S,5}$ and $F_{S,6}$ are placed such that the paths of the face ($F_{S,1}$,$F_{S,2}$,$F_{S,5}$,$F_{S,3}$,$F_{S,4}$,$F_{S,6}$) do not intersect each other (b).

Next, we test whether the paths belonging to the same face of $\mathcal{G}_S$ intersect each other (Figure 4(a)). If so, the artist inserts one or more feature points to avoid intersections between the paths as shown in Figure 4(b).

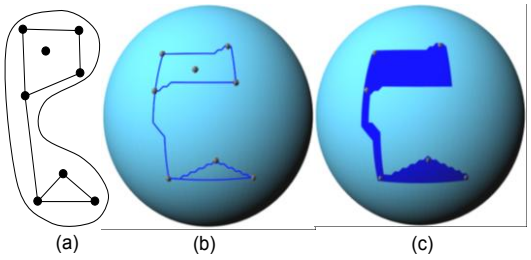

Figure 5. Construction of the seed patch: the graph G_I in the image region (a), the graph G_S on 3D surface (b), selection of triangles using G_S (c).

After constructing $\mathcal{G}_S$, we partition the mesh into regions along the paths of $\mathcal{G}_S$, each region corresponding to a face of $\mathcal{G}_S$ (Figure 5(b)). Next, the seed patch is constructed by merging (1) triangles of inner regions and (2) triangles encountered along the bridging paths (Figure 5(c)).

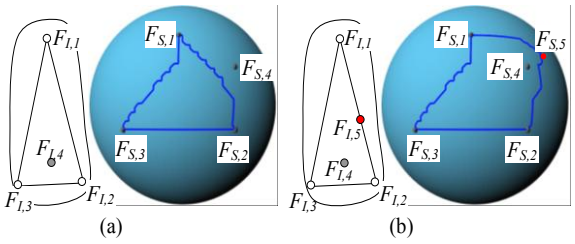

Figure 6. The feature point $F_{S,4}$ is outside the region bounded by ($F_{S,1}$,$F_{S,2}$,$F_{S,3}$) (a); $F_{S,5}$ is added such that $F_{S,4}$ is inside ($F_{S,1}$, $F_{S,2}$,$F_{S,3}$,$F_{S,5}$) (b).

Finally, we need to test if the acquired seed patch contains all the feature points. As previously stated, the graph may contain isolated feature points. Consider a 2D feature point $F_{I,4}$ and its corresponding 3D feature point $F_{S,4}$ as shown in Figure 6. Since $F_{I,4}$ is located inside a face, $F_{S,4}$

should be located inside the corresponding mesh region. If not, the artist is required to insert additional feature points as shown in Figure 6(b) so that both $F_{I,4}$ and $F_{S,4}$ are inside the face and the mesh region, respectively.

It is worth noting that a set of edges that do not form any face may self-intersect when embedded onto the 3D surface. This is useful for creating a self-overlapping texture on the mesh of the 3D model as shown in Figure 7. Mesh vertices located where the texture self-overlaps have several texture coordinates, each of which corresponding to a vertex in the patch.

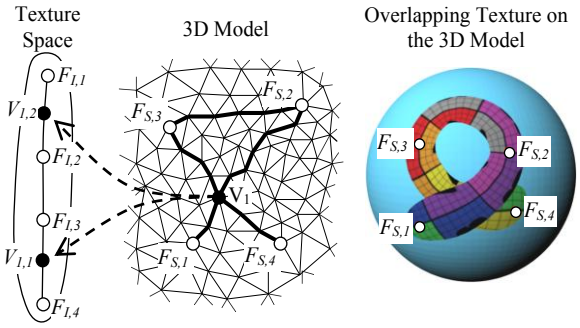

Figure 7. Self-intersecting paths on the 3D model: the vertex V_1 located at the crossing of the paths ($F_{S,1}$,$F_{S,2}$) and ($F_{S,3}$,$F_{S,4}$) on the 3D model has two corresponding vertices $V_{1,1}$ and $V_{1,2}$ in the patch.

4.3. Initial parameterization of the seed patch

Now that the seed patch has been created, we compute the texture coordinates of every vertex (Figure 8). Texture coordinates are computed with different methods, depending on whether the vertex is a feature point, belonging to a path connecting a pair of feature points or inside a face of the feature-point graph.

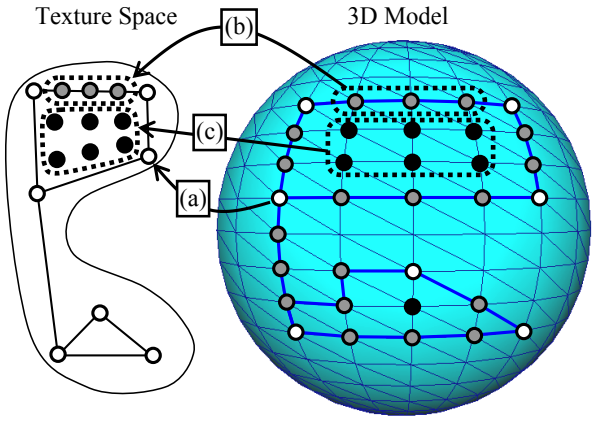

Figure 8. Initial parameterization of the seed patch.

Vertices corresponding to feature points are placed at the user-specified positions in the texture space (shown by the arrow (*a)* in Figure 8). Next, the vertices belonging to the paths of $\mathcal{G}_S$ are uniformly distributed along the corresponding edges of $\mathcal{G}_I$ (arrow (*b)* in Figure 8). Finally, other vertices belonging to a face of $\mathcal{G}_S$ are placed in the interior of the corresponding face of $\mathcal{G}_I$ (arrow (*c)* in Figure 8). Since the faces of $\mathcal{G}_I$ are convex, a simple parameterization method [Flo03b] can be used, which assumes the boundary of the parameterization to be fixed and convex. The initial parameterization of the seed patch may exhibit a high degree of distortion, but a precise parameterization is not critical to the final map, since the mapping will be modified in the subsequent steps.

5. GROWING THE PATCH

Our goal now is to find the complete set of triangles to be texture-mapped, along with their parameterization. These are achieved by growing the seed patch with its neighboring triangles and computing its parameterization in the texture space (see Figure 9).

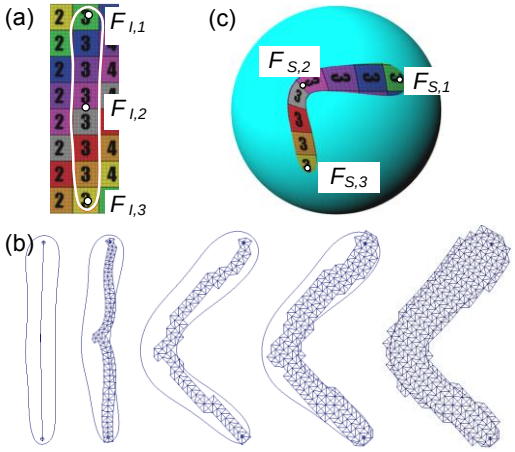

Figure 9. Warping of the image region: image region with the feature points (a); growing of the patch (b), the final model (c).

Note that the structure of the patch and its parameterization are inter-dependent: modifying the structure of the patch requires re-computing its parameterization. Likewise, if the parameterization changes, one needs to modify the structure. For instance, if any of the triangles lie outside the image region after re-computing the parameterization, they must be removed from the patch. Our proposed method adopts an incremental approach: At each iteration, the current patch is recomputed for its parameterization (Section 5.1), followed by an update of its structure (Section 5.2). This is repeated until the patch completely covers the given image region.

5.1. Parameterization of the patch

An unconstrained planar embedding of the patch is first computed using a conventional method. The boundary of the texture region may have any shape; therefore, we use the free-boundary conformal parameterization proposed by [Lev02a], because of its low computation time. Since the patch is grown incrementally, we compute the conformal parameterization using an iterative solver. Such an approach enables us compute the final parameterization with successive approximations starting from an initial parameterization of the patch; in addition, the iterations can be stopped whenever necessary. These features are required for the algorithm to grow the patch (see subsection 5.2.1)

Next we align the feature points of the patch with those of the image region, which can be achieved by warping either the planar embedding of the patch or the image region. In our work, we have chosen to warp the image region whose boundary polygon usually contains many less vertices than the patch (Figure 9).

We compute the warping using a method similar to the one proposed by [Seo10a] We construct a continuous 2D time-dependent vector field **v**(x,y,t) and obtain the new positions of a vertex *p* of the image region by applying a pathline integration of **v**(x,y,t) starting from *p*. Intuitively speaking, the vector field **v**(x,y,t) defines a 2D time-varying velocity vector for all points of the 2D space and for an interval of time. The new position of a point is obtained by moving it according to the velocity specified by the vector field at the position of the point during a time interval.

This vector-field based deformation approach is motivated by two observations: First, foldover-free warping is guaranteed, due to the fact that pathlines of vector field do not intersect in the 4D space-time domain, which is a well-known property of vector fields [Von06a]. Second, the deformation is continuous. Since the vector field **v**(x,y,t) is continuous, so is its integral.

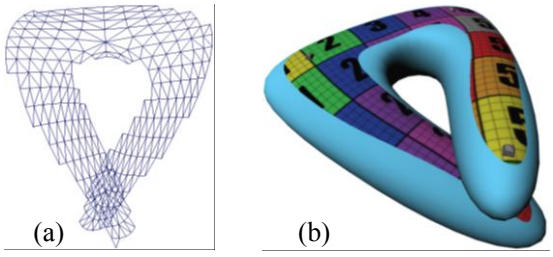

Figure 10. A patch (b) whose planar embedding self-overlaps (a).

One limitation of this method is that it does not allow overlap between different parts of the image region; instead, the method produces highly distorted deformation as shown in Figure 10(b). Allowing overlaps in the warping is necessary in case the planar embedding contains overlapping parts as shown in Figure 10. We show later in this section how to overcome this problem by defining regions of influence for the feature points, allowing the overlapping between different parts of the image region.

5.1.1. Vector Field Based Warping

We warp the image region such that its feature points are moved from their original position (source position) to the position of the corresponding feature points of the patch (target position). The key idea is to relate the movement of the points of the image

region to the trajectory of the feature points moving from the source to the target positions; We define a vector field function **v***(x,y,t)* that relates the instantaneous velocity of the points to those of the feature points along their trajectory. Hereafter, we denote F_i(t) and $\Delta F_i(t)$ the position and velocity of the feature point *i* respectively for $t \in [0,1]$; $F_i(0)$ and $F_i(1)$ represent the source and target positions respectively. The position F_i(t) is obtained from a linear interpolation of $F_i(0)$ and $F_i(1)$.

Given a pair of feature points *i* and *j* and a point *p(t)* in the image region, we compute the relative coordinates $x_{p,i,j}$ and $y_{p,i,j}$ of *p(t)* in the local coordinate frame defined by $F_i(t)$ and $F_j(t)$ (Figure 11):

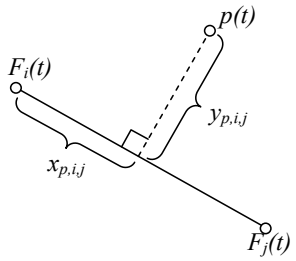

Figure 11. p(t) in the local coordinate frame defined by $F_i(t)$ and $F_j(t)$.

Given the position change of the pair of feature points (*i*, *j*), equation (2) provides the corresponding position change of the point *p(t)*. Note that this equation defines a transformation of *p(t)* which includes rotation, translation and uniform scaling only.

$$p(t) = F_i(t) + x_{p,i,j}\left(F_j(t) - F_i(t)\right) + y_{p,i,j} R_{90}\left(F_j(t) - F_i(t)\right) \quad (2)$$

$$\text{with } R_{90} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Since the number of feature points in the image region is usually more than two, the position of a point is affected by multiple pairs of feature points. Rather than taking into account all possible pairs of

feature point, we use only those pairs whose line segments are completely inside the image region. This guides the deformation to better reflect the shape of the image region.

The total velocity of the point *p(t)* is expressed as a weighted combination of the velocities associated with each pair of feature points. We define a weight for each feature point pair (*i,j*) so that it increases as the point becomes closer to either of the feature points:

$$\beta_{p,i,j} = \frac{1}{\left(d_{p,i} \cdot d_{p,j}\right)^{2\alpha}}$$

$d_{p,i}$ and $d_{p,j}$ are distances between *p(t)* and each of the feature points *i* and *j*. The exponent α determines the smoothness of the warping. We found that the value α=1 gives visually pleasing deformations. These weights are normalized such that their sum is unity for each point *p(t)*.

The function **v**(*x,y,t*) that relates the instantaneous velocity of point *p(t)* with those of the feature points is given by:

$$\mathbf{v}\left(p(t),t\right) = \sum_{i,j}\left(\beta_{p,i,j}\alpha_{p,i,j}\left(\frac{d}{dt}F_i(t) + x_{p,i,j}\left(\frac{d}{dt}F_j(t) - \frac{d}{dt}F_i(t)\right) + y_{p,i,j}R_{90}\left(\frac{d}{dt}F_j(t) - \frac{d}{dt}F_i(t)\right)\right)\right) \quad (3)$$

The purpose of weights $\alpha_{p,i,j}$ is to allow overlaps in warping (see Section 5.1.2). The new position of a point p_0 is obtained by applying a pathline integration of **v**(*x,y,t*) starting form p_0:

$$\begin{cases} \dfrac{dp(t)}{dt} = v\left(p(t),t\right) \text{ for } t \in [0,1] \\ p(0) = p_0 \end{cases}$$

In our current implementation, we use the explicit Euler integration with a constant step size t_s. One integration step involves computing the intermediate solution *p(t)* and updating the values $\beta_{p,i,j}$, $y_{p,i,j}$, and $x_{p,i,j}$ of **v**(*x,y,t*) by using *p(t)* and the feature point positions $F_i(t)$.

5.1.2. Allowing overlaps in the warping:

A distinctive feature of our vector-field based warping is that it deforms the entire 2D space the image region lies in, without regard to the shape of the image region. Consider two distinctive points p_1 and p_2 inside an image region as shown in Figure 11(a). As feature points F_1 and F_4 are designated to become close to each other (Figure 11(b)), the warping produces a highly distorted deformation, which is undesirable. The reason for such artifact is that the positions of the points are affected by all the feature points, regardless of the shape of image region. Allowing overlaps in the image region would be a better alternative, since it reduces such distortion as shown in Figure 12(c).

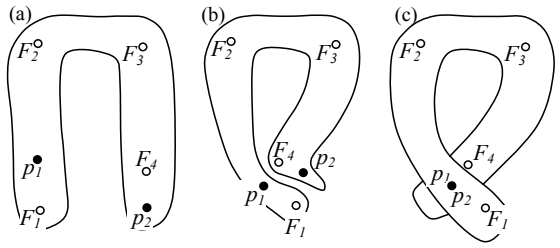

Figure 12. Overlapping of the image region: the image region (a) is deformed such that feature points F_1 and F_2 come close to each other; deformation produced by the vector-field based warping (b); overlapping is acquired by reducing the influence of F_4 on p_1 and F_1 on p_2, respectively.

Overlaps are implemented by restricting the influence of the feature-point pairs to their neighboring area inside the image region. We cluster feature points into groups and divide the image region into segments, each associated with a group. Since the influence of each group of feature points is restricted to a segment, overlap can occur between different segments of the image region.

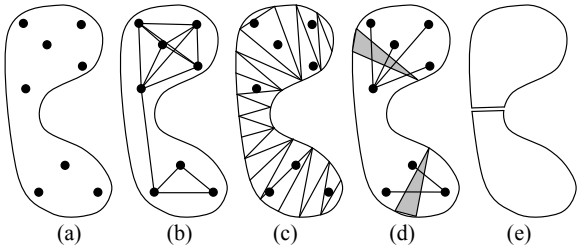

Figure 13. Clustering of the feature point: the boundary of the image region along with feature points provided (a), segments of feature points that do not intersect the boundary (b), constrained Delaunay triangulation (c), clustering of feature points (d), region segments (e).

There are three steps involved in construct the feature-point groups: (1) we first compute the constrained Delaunay triangulation of the image region (Figure 13(c)). (2) We then search for a triangle intersecting with the maximum number of line segments connecting pairs of feature points (Figure 13(d)). (3) The feature points whose line-segment intersects the triangle and which are not already assigned to an existing group are added to a new group. We repeat the process of finding the next triangle with the largest number of intersecting line segments and making a new feature-point group, until every feature point is assigned to a group.

Once all feature points have been assigned to a group, we compute the influence area of each group. For each vertex *p* of the image region, we define a weight $\alpha_{p,l}$ associated with the influence area group *l*

on *p*. We compute these weights individually for each group *l* by minimizing the following function:

$$\min_{\alpha_{p,l}} \sum_{p,q \in E} \left| \alpha_{p,l} - \alpha_{q,l} \right|^2 \qquad (2)$$

Where *E* is the set of adjacent vertices of the constrained Delaunay triangulation such that $p,q \in E$ if either vertex *p* or vertex *q* (or both) are not coincident with a feature point. Coefficient of a feature point is set to either 1 or 0, depending on whether it belongs to group *l* or not.

We use the above computed weights to segment the image region as shown in Figure 13(e); the segmentation is required for the triangulation of the image region (see Section 5.2.1).

Then, for each vertex *p*, we compute the influence weight $\alpha_{p,i,j}$ for each pair of feature points (*i, j*) by summing the weights $\alpha_{p,l}$ of all the groups that contain at least one of the feature points *i* or *j* of the pair:

$$\alpha_{p,i,j} = \sum_{i,j \in l} \alpha_{p,l}$$

These weights $\alpha_{p,i,j}$ are then normalized such that their sum is unity for each vertex *p* of image region. Once computed, these weights are integrated into the equation (3) of the warping. The values $\alpha_{p,i,j}$ continuously change across the vertices of the constrained Delaunay triangulation of the image region, yielding a smooth-looking deformation.

5.1.3. Limitations of the vector field based warping

The warping method does not work in two cases. The first case arises when two feature points belonging to the same pair have same position at the same time during the warping of the image; the local coordinate frame shown in Figure 11 cannot be defined if $F_i(t)$ and $F_j(t)$ are coincident.

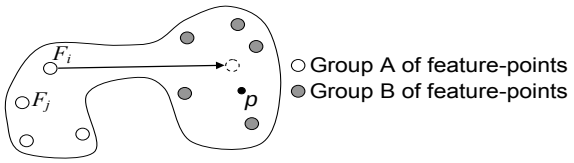

Figure 14. Case for which the warping produces a foldover: a feature point Fi is moved inside the image region to a vertex p whose weigh value $\alpha_{p,i,j}$ is very small.

The second case happens when a pair of feature points (F_i,F_j) is moved inside the image region close to a point *p* whose influence weight $\alpha_{p,i,j}$ is very small.

5.2. Updating the patch structure

As previously stated, we grow the patch iteratively until its planar embedding covers the image region; one iteration consists of computing a parameterization of the patch combined with the warping of the image region, and updating the patch structure. Hereafter, we denote P_{Prev} and P_{Curr} the position vectors of patch vertices corresponding to the parameterization computed at the previous and current iteration respectively. The patch structure is updated as follows: (1) We remove patch triangles that are located outside the image region. Note that although all triangles of the seed patch are initially inside the image region, they may leave the region as we continuously update their parameterization. This is typically encountered when there is a high degree of distortion in the initial mapping of the seed patch, due to the fact that we roughly approximate the initial placement of the seed patch on the surface using geodesic paths without considering its texture mapping distortion. (2) We add triangles that are adjacent to the patch boundary and inside the image region.

In order to obtain a final patch whose surface is topologically equivalent to a disc, we maintain the disc-like topology of the seed patch through all the updates of the patch structure. In particular, we should avoid the patch splitting into more than one as we remove triangles from it. We describe how we avoid such topological change in what follows.

5.2.1. Removing triangles from the patch

In order to maintain the topology of the patch, we test if the intersecting part of the patch with the image region is composed of several disconnected components as illustrated in Figure 15(b) (step (a) in the flowchart of Figure 19). If so, the vertex positions are rolled back to their positions at the previous iteration P_{Prev} where the whole patch was inside the image region (Figure 15(a)). We then compute an approximated parameterization corresponding to the intermediate positions of the vertices between the previous positions P_{Prev} and the positions corresponding to the *final* parameterization (step (b) in Figure 19). The approximated parameterization is computed such that the part of the patch intersecting the image region forms one component (Figure 15(c)).

Rather than computing the exact intersection of the patch with the image region, we find a set *S* of connected vertices from the patch, located inside the image region and containing at least one feature point. If *S* also contains all other feature points, the intersecting part of the patch with the image region can be considered as one component. In order to construct *S*, we first compute a triangulation of the image region by triangulating each of its segments (the algorithm to segment the image region is given in section 5.1.2).

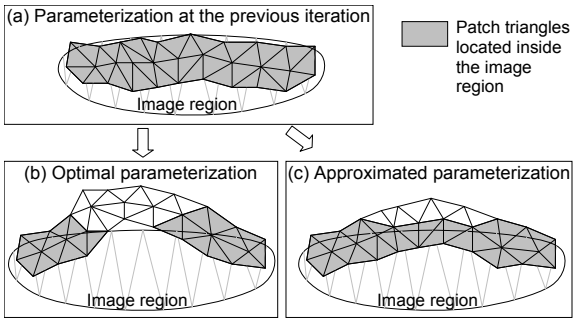

Figure 15. Computing an approximated parameterization.

The algorithm to construct S works by keeping for each vertex v_i of the set S, the triangle T_i of the image region that contains v_i. Initially, S is composed of one feature point; feature points are always inside the image region and we can easily find the triangle of the image region that contains them. The set S is then enlarged iteratively by visiting the edges (v_i,v_j) whose vertex v_i is in the set and v_j outside. Since we know the triangle T_i of the image region that contains v_i, we compute the intersections of (v_i,v_j) with T_i and its neighboring triangles until no more intersecting triangles are found or the edge (v_i,v_j) has crossed the boundary of the image region. If (v_i,v_j) does not intersect the boundary region, v_j is added to S and the triangle T_j that contains v_j is the last triangle intersecting the edge (v_i,v_j). Note that the algorithm to construct S works even when the image region self-overlaps because the test to determine if a patch vertex is inside/outside the image region only requires computing intersections locally with the triangles of the image region.

Once S is constructed, we remove patch triangles whose three vertices do not belong to S (steps (c) and (d) in Figure 19).

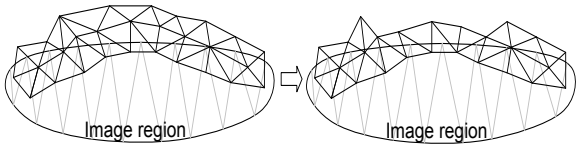

Figure 16. Removing patch triangles whose three vertices are outside the image region.

5.2.2 Insertion of triangles to the patch

There are two different cases when adding triangles to the patch: In the first case, a final parameterization has been computed; we grow the patch by adding a triangle strip along its boundary and inside the image region as shown in Figure 17(a) (step (e) in Figure 19). In the second case, only an approximated parameterization has been calculated; triangles need to be added so that a final parameterization can be computed and the part of the patch intersecting the image region forms one component (step (f) in Figure 19). To achieve this, we find all edges that are shared by two triangles and whose two vertices are on the boundary of the patch (edge e in Figure 17(b)). We then add triangles to the endpoint of the edge which is inside the image region.

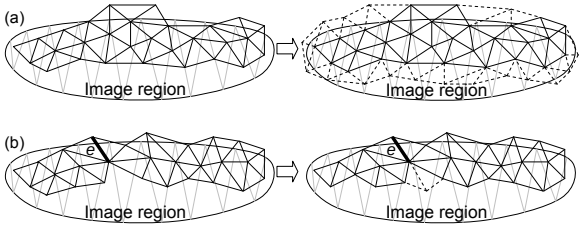

Figure 17. Insertion of triangles.

During the process of adding triangles, we must ensure that the local connectivity of the patch is kept consistent with that of the mesh; given a vertex p of the patch and its corresponding vertex m on the mesh, every vertex connected to p must have one corresponding vertex among those connected to m. The consistency of the local connectivity is broken when two patch vertices corresponding to the same mesh vertex, are adjacent to the same patch vertex (Figure 18(a)). As mentioned in Section 4.2, mesh vertices located inside a self-overlapping texture region (Figure 7) typically have several texture coordinates. To maintain the consistency between the patch and the mesh, we merge the patch vertices corresponding to the same mesh vertex whenever they become adjacent to the same patch vertex (Figure 18(b)).

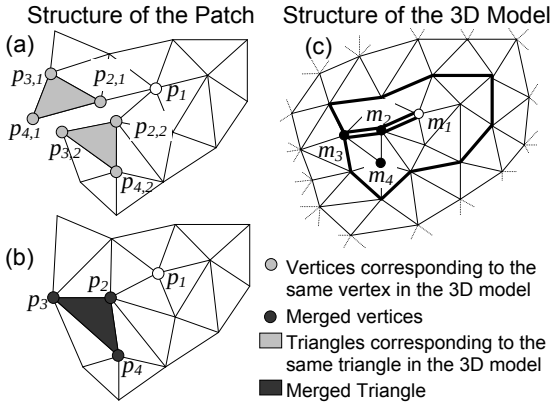

Figure 18. Consider two patch vertices $p_{2,1}$ and $p_{2,2}$ corresponding to the same vertex m_2, and their adjacent vertex p_1 corresponding to m_1. Clearly, the local connectivity around p_1 shown in (a) and the one around m_1 shown in (b) are inconsistent. We merge $p_{2,1}$ and $p_{2,2}$ into p_2 to correct this problem.

We give the flowchart of the algorithm to grow the patch.

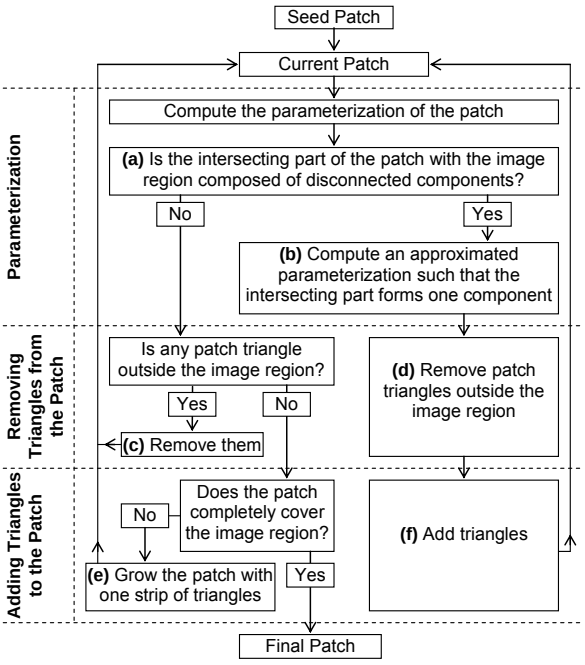

Figure 19. A flowchart of the process of growing the patch.

6. UPDATING THE TEXTURE OF THE 3D MODEL

Once the construction of the patch has been completed, the pixels of image region are transferred into the texture of the 3D model; this implies that the 3D model has already a texture with a global parameterization.

We first warp the image region using vector-field based warping (section 5.1.1) as shown in Figure 20(c). We then update the texture of the 3D model by transferring the image pixels inside the patch triangles into the texture (Figure 20(d)). Note that several images can be mapped successively on the same 3D model to create composition of textures.

Another interesting feature of our approach is to create self-overlapping textures on the 3D model as illustrated in Figure 7. Triangles of the 3D model located in the overlapping parts are textured several times with different portions of the image, each portion corresponding to a triangle in the patch. To determine the hiding-and-hidden relations among the overlapping parts of the texture, we use a technique similar to the painter's algorithm; we sort all the patch triangles by their depths and process them in this order.

The depth value of the patch triangles connected to a feature point is set to an index provided by the artist. Depth values ω_i of other triangles are then computed by minimizing the following function:

$$\min_{\omega} \sum_{(i,j)\in E} \left|\omega_i - \omega_j\right|^2$$

where E is the set of adjacent triangles such that $(i,j) \in E$ if either triangle i or j (or both) are not connected to a feature point.

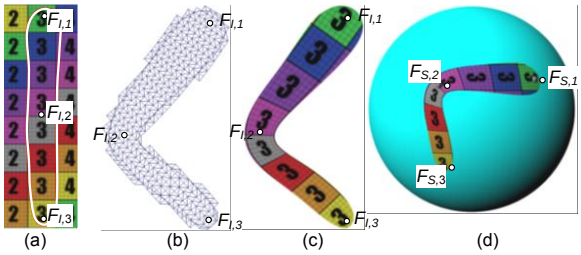

Figure 20. Transferring the pixels of the image region into the texture of the 3D model: image region (a), final patch in the texture space (b), warped image region (c), and 3D model with the updated texture (d).

7. RESULTS

Our texturing tool has been implemented as a plug-in to Maya, with which the user can add and edit feature constraints interactively. The computation time for generating the texture mapping ranges from half a second to a few seconds depending on the complexity of the textured 3D models and the size of the image region.

Our method is demonstrated with several examples corresponding to different cases of texture mapping, showing its versatility. The example shown in Figure 1 and the last one in Figure 23 demonstrate the texture mapping of surfaces with sharp features. The first example in Figure 23 shows how to create overlapping textures on 3D models. The two next examples show the texture mapping with large displacements of the feature points on the 3D model.

7.1. Comparison with previous work

We have compared our method with those proposed by Sun et al. [Sun13a] and Schmidt et al. [Sch06a]. In order to provide a qualitative comparison, we have mapped a same texture onto a same surface using the three different methods (Figure 21). The method by [Sch06a] is clearly the one generating a texture mapping with the highest distortion. [Sun13a] had already mentioned that their method performs better than that of [Sch06a].

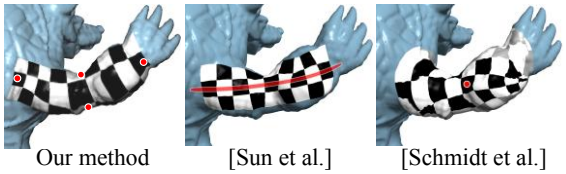

Figure 21. Comparison of our method with the methods proposed by [Sun13a] and by [Sch06a].

Figure 22 shows the texture coordinates generated by our method and by [Sun13a]. In case of [Sun13a], the mapping distortion is high for triangles along the

boundary of the textured area. In case of our method, the mapping distortion is evenly distributed over the textured area.

In order to make a quantitative comparison, we have computed the L^2 stretch metric which measures the deformation of the mapping; the formula of the L^2 stretch is given by [San01a]. As shown in Figure 22, the value of the L^2 metric for our method is significantly lower than that of the method by [Sun13a].

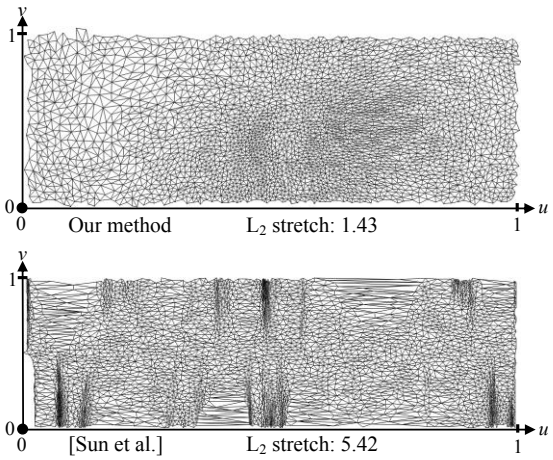

Figure 22. Comparison of the texture coordinates generated by our method with those generated by the method of [Sun13a]. These texture coordinates are those of the surface shown in Figure 21.

7.2. Limitations

Our method has several limitations. As mentioned in section 5.1.3, the proposed warping method may not work when feature points in the warping have same position. Our method fails when the planar embedding of the patch contains triangle flips. However, in practice, we found that triangle flips rarely occur. Finally, since our method is based on conformal parameterization, the textured region tends to grow substantially on surfaces with sharp features; this is because the conformal parameterization minimizes the angular distortion and does not preserve the distances across the surface [Lev02a]. In some cases these undesirable artifacts can be avoided by providing additional feature points. The removal of these artifacts is not always possible. One example is shown in the third row of Figure 23; the level of distortion is so high that it cannot be reduced by placing more feature points.

8. CONCLUSION

We have proposed a method for local parameterization applied for the texture-mapping of images on triangular meshes. Compared to previous work on local parameterization, our method allows multiple feature constraints in the form of correspondence between points in the texture and vertices on the 3D model.

9. REFERENCES

[Car04a] Carr, N. A., and Hart, J. C. 2004. Painting detail. In Proceedings of SIGGRAPH 2004, 842–849.

[Des02a] Desbrun, M., Meyer, M., and Alliez, P. 2002. Intrinsic parameterizations of surface meshes. In Proceedings of Eurographics, 2002.

[Eck01b] Eckstein, I., Surazhsky, V., and Gotsman, C. 2001. Texture mapping with hard constraints. Computer Graphics Forum 20, 3.

[Flo03a] Floater, M., and Hormann, K. 2003. Recent advances in surface parameterization. Multiresolution in Geometric Modelling Workshop.

[Flo03b] Floater, M. 2003. Mean value coordinates. CAGD 20, 1, 19–27.

[Hur99a] Hurtado F., Noy M., and Urrutia J., Flipping edges in triangulations. Discrete Comput. Geom.,22(3):333–346, 1999.

[Kra03a] Kraevoy, V., Sheffer, A., and Gotsman, C. 2003. Matchmaker: constructing constrained texture maps. In Proceedings of SIGGRAPH 2003, 326–333.

[Lef05a] Lefebvre, S., Hornus, S., and Neyret, F. 2005. Texture sprites: Texture elements splatted on surfaces. In ACM SIGGRAPH Symposium on Interactive 3D Graphics (I3D).

[Lev02a] Levy, B., Petitjean, S., Ray, N., and Mallet, J.-L. 2002. Least squares conformal maps for automatic texture atlas generation. In Proceedings of SIGGRAPH 2002, 362–371.

[Lev01b] Levy, B. 2001. Constrained texture mapping for polygonal meshes. In Proceedings of SIGGRAPH 2001, 417–424.

[May05a] Maya, http://www.alias.com/.

[Mey02a] Meyer, M., Lee, H., Barr, A., and Desbrun, M. 2002. Generalized barycentric coordinates on irregular polygons. J. Graph. Tools 7, 1, 13–22.

[Pra00a] Praun, E., Finkelstein, A., and Hoppe, H. 2000. Lapped textures. In Proceedings of SIGGRAPH 2000, 465–470.

[San01a] Sander, P. V., Snyder, J., Gortler, S. J., and Hoppe, H. 2001. Texture mapping progressive meshes. In Proceedings of SIGGRAPH 2001, 409–416.

[Sch06a] Schmidt R., Grimm C., Wyvill B.: Interactive decal compositing with discrete

exponential maps. In Proceedings of SIGGRAPH 2006, 25(3): 605–613.

[Seo10a] Seo, H., Cordier F.: Constrained Texture Mapping using Image Warping. Comput. Graph. Forum 29(1): 160-174 (2010)

[Sol02a] Soler, C., Cani, M.-P., and Angelidis, A. 2002. Hierarchical pattern mapping. In Proceedings of SIGGRAPH 2002, 673–680.

[Sun13a] Sun Q., Zhang L., Zhang M., Ying X., Xin S.-Q., Xia J., and He Y., Texture brush: an interactive surface texturing interface. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '13), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, 153-160

[Tur01a] Turk, G. 2001. Texture synthesis on surfaces. In Proceedings of SIGGRAPH 2001, 347–354.

[Von06a] Von Funck W., Theisel H., Seidel H.-P.: Vector field based shape deformations. ACM Trans. Graph. 25(3): 1118–1125 (2006)

[Wei01a] Wei, L., and Levoy, M. 2001. Texture synthesis over arbitrary manifold surfaces. In Proceedings of SIGGRAPH 2001, 355–360.

[Zha05a] Zhang, E., Mischaikow, K., and Turk, G. 2005. Feature-based surface parameterization and texture mapping. ACM Trans. Graphics 24, 1, 1–27.

[Zho04a] Zhou, K., Snyder, J., Guo, B., and Shum, H.-Y. 2004. Iso-charts: Stretchdriven mesh parameterization using spectral analysis. In Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing, 47–56.

[Zho05b] Zhou, K., Wang, X., Tong, Y., Desbrun, M., Guo, B., and Shum, H.-Y. 2005. TextureMontage: Seamless Texturing of Arbitrary Surfaces From Multiple Images. In Proceedings of SIGGRAPH 2005, 1148–1155

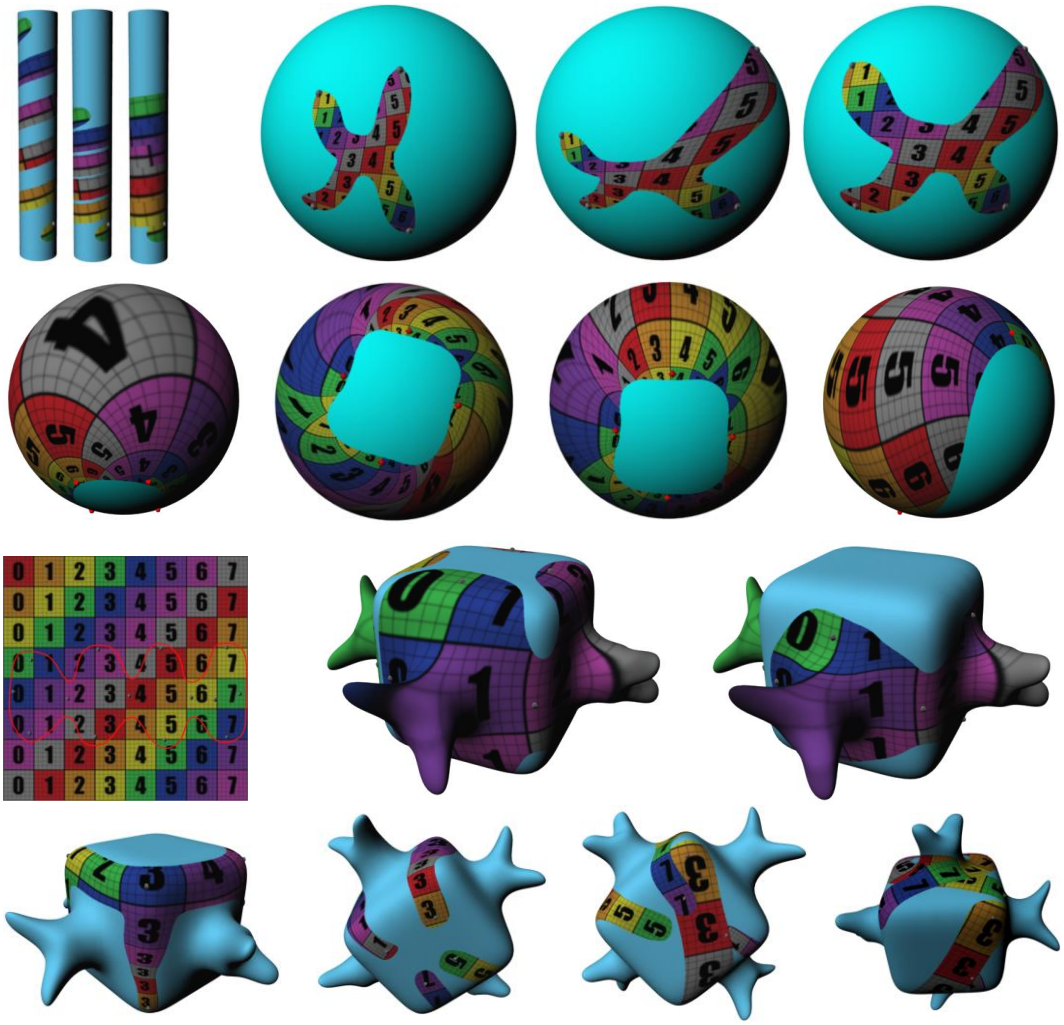

Figure 23. Textured Models.