TextureMontage: Seamless Texturing of Arbitrary Surfaces From Multiple Images

Kun Zhou* Xi Wang* Yiying Tong† Mathieu Desbrun† Baining Guo* Heung-Yeung Shum*

*Microsoft Research Asia †Caltech

Abstract

We propose a technique, called *TextureMontage*, to seamlessly map a patchwork of texture images onto an arbitrary 3D model. A texture atlas can be created through the specification of a set of correspondences between the model and any number of texture images. First, our technique automatically partitions the mesh and the images, driven solely by the choice of feature correspondences. Most charts will then be parameterized over their corresponding image planes through the minimization of a distortion metric based on both geometric distortion and texture mismatch across patch boundaries and images. Lastly, a surface texture inpainting technique is used to fill in the remaining charts of the surface with no corresponding texture patches. The resulting texture mapping satisfies the (sparse or dense) user-specified constraints while minimizing the distortion of the texture images and ensuring a smooth transition across the boundaries of different mesh patches.

Keywords: Texture Mapping, Parametrization, Content-based Metric, Geometry-based Metric.

1 Introduction

Texture mapping has long been used in computer graphics as a way to enhance the visual richness of a 3D surface, be it for overly simplified character meshes in game engines or for complex digital models in computer-generated feature films. Seamlessly mapping synthesized and/or real-life textures onto 3D models with little visual distortion can be, however, painstaking to do by hand. As a consequence, multiple techniques have been proposed over the past few years to simplify this meticulous process.

1.1 Related Work

To provide assistance in decorating 3D models, a variety of tools has been developed in the field of digital geometry processing.

Parameterization Without Feature Correspondence Texture mapping of a surface patch is specified through a parameterization, i.e., a one-to-one map from a texture domain to the surface patch. In most cases, a disk-like patch of a non-flat, piecewise-linear surface is mapped onto the plane with inevitable distortion. A first series of tools proposed variational approaches to reduce a particular distortion measure such as angle or area distortion (see, e.g., [Maillot et al. 1993; Hormann and Greiner 1999; Sander et al. 2001; Lévy et al. 2002; Desbrun et al. 2002; Floater 2003] and the survey by Floater and Hormann [2003]). The content of the texture can even be taken into account to create a signal-optimized parameterization [Sander et al. 2002; Balmelli et al. 2002]. However, if one has to map a whole object instead of a single patch, a partitioning (or *atlas of charts* [Grimm and Hughes 1995]) of the surface into genus-0 patches must be completed first. Here again, various tools have been designed to offer an automatic or user-guided atlas creation [Maillot et al. 1993; Eck et al. 1995; Lee et al. 1998; Lévy et al. 2002; Zhou et al. 2004; Zhang et al. 2005], even with smooth transition of texture coordinates across patch boundaries [Khodakovsky et al. 2003]. Recently, variants have been introduced for objects of genus-0 through spherical embedding [Gu and Yau 2003; Gotsman et al. 2003; Praun and Hoppe 2003] or higher genus models through a single cut [Sheffer and Hart 2002; Gu et al. 2003] where an atlas is no longer needed.

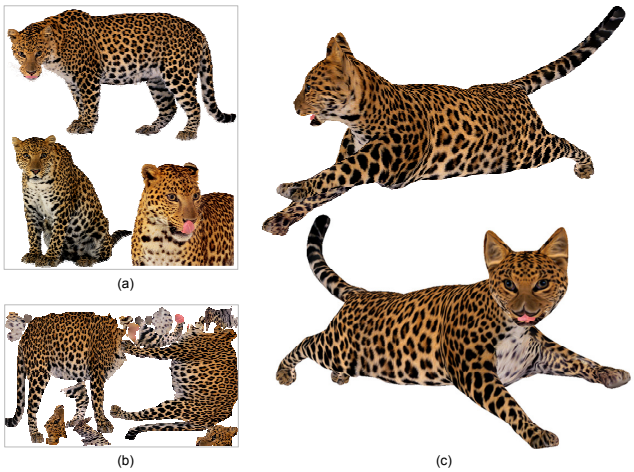


Figure 1: *TextureMontage: from a cat model and some images of a leopard (a), a user can transfer the leopard's skin onto the 3D model through the definition of an arbitrary number of feature correspondences between the cat and any of the images. A texture atlas is automatically generated (b), resulting in a seamless texturing on the input model (c).*

Parameterization With Feature Correspondence Even equipped with these tools, decorating surfaces often requires *feature correspondences*: specific features of the surface often need to be mapped to a particular region of the texture. For instance, decorating a humanoid model will require the texture representing the eyes to be mapped to the actual eyes' position on the mesh. This difficult problem found practical solutions in [Lévy 2001] where soft constraints were implemented, as well as in [Desbrun et al. 2002] through the use of Lagrange multipliers and in [Eckstein et al. 2001] where Steiner vertices were inserted to satisfy hard constraints. A recent algorithm, called *Matchmaker* [Kraevoy et al. 2003] provides a more robust solution by automatically partitioning a mesh into genus-0 patches, and allowing the user to set correspondences between the patches and one (or two) texture image(s). C^1 continuity through patch boundaries can even be obtained, but this procedure is time intensive for the user as a large number of correspondences needs to be defined. Note that two recent algorithms, cross-parameterization [Kraevoy and Sheffer 2004] and inter-surface mapping [Schreiner et al. 2004] propose a similar approach for the design of mappings *between two surfaces* instead of

*e-mail: {kunzhou,t-xiwang,bainguo,hshum}@microsoft.com

†e-mail: {yiying,mathieu}@caltech.edu

between a surface and texture space, and can also handle feature correspondences. Although our method is algorithmically similar to these last two methods, we construct instead a map *between a surface and several texture images*.

Texture Painting and Synthesis Painting directly on a surface is a very convenient tool for artists to help design textures on 3D models [Igarashi and Cosgrove 2001; Carr and Hart 2004]. The process is, however, often laborious and requires artistic skills. On the other hand, texture synthesis (see, i.e., [Praun et al. 2000; Turk 2001; Wei and Levoy 2001; Soler et al. 2002]) can semi-automatically decorate objects if *repeating* textures are desired, but is very limited in the style of design an artist can obtain otherwise.

Texture Mapping in 3D Scanning The issue addressed in this paper is also related to the texture mapping problem in 3D scanning. In this context, the textures are captured directly from the object being texture mapped. The correspondence between the object and images is thus automatically *implied* by the scanning process, and only distortions due to the camera have to be accounted for in fitting the images to the object. Therefore, our approach can be seen as generalization of existing methods [Rocchini et al. 1999; Neugebauer and Klein 1999].

1.2 Motivation

In light of this overview of existing tools, it appears that making use of *multiple texture images* for surface decoration is still a labored task. Indeed, seamlessly mapping a series of real-life photos onto a 3D model (such as in Figure 1) requires the features of the texture images and the models to be aligned, demanding an impractical amount of manual work to put dense constraints along patch boundaries. Additionally, even if boundary continuity can somehow be ensured automatically, the number of correspondences required to guarantee that *each* triangle of the original domain has corresponding texture coordinates in a texture image can be arbitrarily large for meshes of high genus, making it once again impractical for the user.

However, using multiple texture images could significantly reduce the amount of time needed for 3D model decoration. Imagine for instance an artist trying to decorate a model from a series of digital pictures or man-made drawings (of a given subject under various points of view, or of different subjects): combining parts of these images in order to create a "composite" texture, without having to care about cutting the model in patches or painstakingly ensuring continuity across textured regions, would render the task as easy as a *photo montage* [Agarwala et al. 2004]. Consequently, we propose in this paper to provide a digital photomontage-like decoration of meshes—although the algorithms involved are seemingly more difficult due to the non-preventable mapping distortion between a surface and its local parameterization texture.

1.3 Contributions

Our new technique, called *TextureMontage*, offers a powerful, practical tool for decorating arbitrary 3D models using multiple texture images. The user specifies an (arbitrary large or small) set of feature correspondences, each constraining a vertex on the mesh to a point in one or more images (see Figure 2). Given these user-specified correspondences, the algorithm then computes, through optimization, the texture coordinates of most of the surface triangles taking into account both mapping distortion and texture color continuity, to result in a visually seamless decoration of the mesh. Finally, for the few mesh regions that do not get automatically mapped to a texture image (by lack of user-input correspondences), an automatic

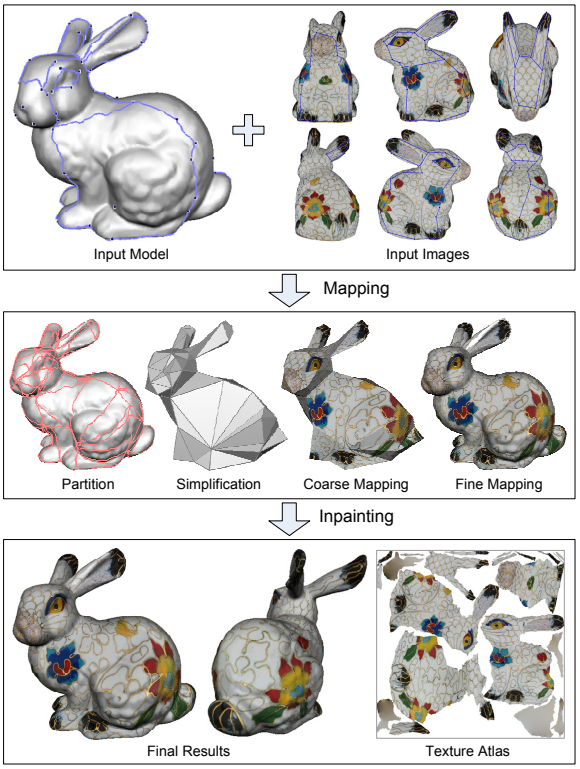


Figure 2: *Decoration process of TextureMontage: from a 3D model and a few images, an initial mapping of feature correspondences is input by the user; then a hierarchical optimization procedure is run to compute most of the mapping; finally, the resulting parts of the mesh not yet textured are inpainted to complete the texture atlas.*

surface texture inpainting technique using a Poisson-based interpolation [Pérez et al. 2003] is used to fill in these gaps. The inpainted colors in these regions render the overall mesh decoration seamless.

To achieve this general framework for combining multiple images over arbitrary 3D models with minimum user interaction, we make heavy use of existing algorithms, and add the following contributions:

- An automatic approach to partition a mesh *and* multiple images *simultaneously* according to (a few or multiple) user-specified correspondences (Section 3.1).
- A content-based measure of texture mismatch across patch boundaries and an interleaved texture-coordinate optimization algorithm to minimize it while still optimizing geometric distortion (Section 4).
- A surface texture inpainting technique to smoothly fill in non-textured regions on a surface described in Section 5.

2 Overview and Terminology

As shown in Figure 2, the decoration process in *TextureMontage* progresses as follows:

1. **Partition of Mesh and Images**: After the user has specified feature correspondences, we generate a correspondence-driven partitioning of both the mesh and the images.
2. **Progressive Mesh Creation**: We then use repeated half-edge collapses to create a progressive mesh that preserves the

boundaries of the aforementioned partition; the base mesh allows us to build a coarse mapping between the mesh and images.

3. **Coarse-to-fine Map Construction**: While we unroll the progressive mesh from the base mesh to its original form, we use both geometric and texture information to derive the texture coordinates of each inserted vertex, minimizing both parameterization distortion and texture mismatch along patch boundaries.

4. **Surface Texture Inpainting**: For the triangular patches with no corresponding texture coordinates, we use an automatic or user-guided inpainting to fill in the holes by upsampling the region and coloring each inside vertex in order to smoothly blend in with the surrounding textures.

5. **Atlas Generation**: Finally, we create a texture image for each inpainted hole; the only thing left to do is to pack all the texture patches used into a texture atlas.

Note that our use of a fine-coarse-fine strategy is not arbitrary: even if a parameterization can be computed directly after the first step, we adopt a hierarchical approach as in [Schreiner et al. 2004] to ensure a good non-linear minimization of the texture assignments; our goal of enforcing smooth texture transition across path boundaries further motivates this progressive scheme. Also, notice that we will not give details on the last step of the algorithm since the chartification and parameterization of hole regions and the atlas packing are done as in [Lévy et al. 2002]. Finally, and to reduce possible confusions, we will denote by *path* a polyline (i.e., sequence of vertices) traced on a mesh, while *curve* will be used to describe a polyline (i.e., sequence of 2D coordinates) in the texture domain.

Preprocessing Before starting our algorithm, we perform three preprocessing steps on the input images. First, to deal with possible color mismatch caused by different lighting conditions in the source images, we use Adobe Photoshop to adjust exposure differences and color balances between these images. Then, we interactively segment the useless background regions using, for instance, recent image cutout techniques [Li et al. 2004]. The output for each pixel is an alpha value ranging from zero to one, where zero values are assigned only to background pixels. Finally, for each pixel, we store the distance to the nearest non-background pixel. The preprocessing steps take a few minutes and only need to be carried out once for each image, prior to the texturing process.

3 Texture Map Initialization

As in [Kraevoy et al. 2003], we begin our texturing process by running a partition algorithm that looks for a set of *path-curve* pairs on the mesh and the images. However, we depart from their strategy by *not* requiring that the paths partition the mesh into a set of triangular patches. As a consequence, some patches on the mesh may not have any corresponding texture patches after partition; but we will show, in Section 5, that this issue can be easily resolved through a surface texture inpainting process. Thus, our new strategy allows the user to deal with any model and any feature correspondences. Note that even in the worst case, i.e., when no feature correspondences are specified, our algorithm still proceeds normally with the whole mesh considered as a unique patch.

3.1 Mesh Partitioning by Mesh-Texture Matching

The initial partitioning algorithm proceeds similar to [Kraevoy et al. 2003; Kraevoy and Sheffer 2004; Schreiner et al. 2004]: 1) First, we compute the shortest paths between all pairs of feature vertices, and put them in a priority queue ordered by length; 2) We then test the *shortest* path (i.e., the first on the queue): if both ends of the path have corresponding texture coordinates in the *same* image, and if it is a valid path-curve pair [Kraevoy et al. 2003], then we add the path-curve pair to the correspondence set. Note that a same pair can be stored twice, one for each direction of the path; 3) Repeat Step 2 until the queue is empty.

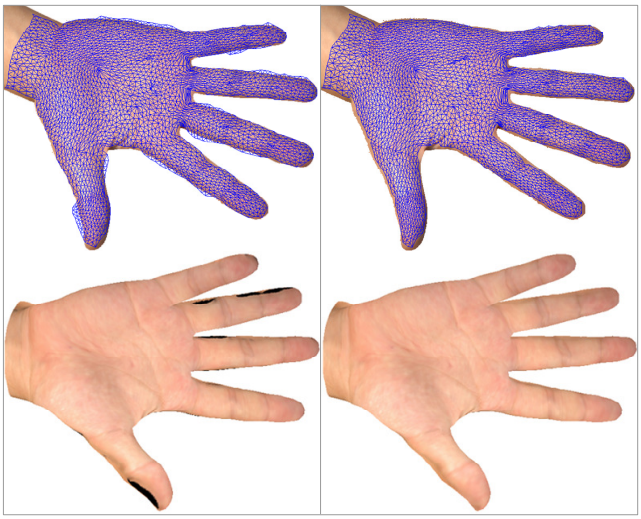
Most of the details of our partitioning algorithm, such as the addition of Steiner points along the paths, are exactly as in [Kraevoy et al. 2003]; there are, however, fundamental differences. First, we use this algorithm to derive an ab-initio partition of the mesh, while the authors of [Kraevoy et al. 2003] started from a pre-cut mesh. Second, we do not make a priori assumptions on the texture content contrary to what was used in [Kraevoy et al. 2003]. Instead, we allow multiple texture images and do not enforce any constraints on the number of feature correspondences. As a consequence, some path-curve pairs can cross background regions, causing undesirable partitions. We, therefore, have recourse to the precomputed distance function to evaluate the validity of the image paths; our simple solution is to compute the average distance between an image path and the foreground region. If the distance is less (resp., more) than a given threshold (2 pixels in our implementation), the edge is declared valid (resp., invalid). We uniformly distribute sampling points along the image paths such that the distance between successive sampling points is less than one pixel. The distance for each sampling point can be computed by interpolating the precomputed distance function on the texture image during the preprocessing stage. Then the distance for the image curve is computed as the average of the distances of all sampling points.

It is important to remark that, with basically no additional effort, our approach supports feature lines and feature polygons: we can simply add the path-curve pairs corresponding to feature lines and polygons to the set of pairs *before* running the path matching algorithm.

3.2 Base Mesh and Coarse Texture Map

Once the set of pairs has been established, we run a half-edge collapse simplification to build a progressive mesh structure. As in [Schreiner et al. 2004], we retain feature vertices and constrain the edge collapse sequence to preserve the topology of both the path network and of the original mesh. The result of this simplification is a *base mesh*, for which each edge corresponds to a path on the original mesh. Then, for each triangle of the base mesh, if it corresponds to a texture triangle in one image, we map it into that image by setting the corresponding texture coordinates for its three vertices (this chart will be parameterized over its corresponding image plane in the next step). The triangles with no corresponding texture triangles are flagged as empty, and will be filled in later (see Section 5). Because we prevent some edges from collapsing in order to preserve topology, the base mesh often ends up containing vertices that are *not* feature vertices; all triangles containing those vertices are also flagged as empty for now.

For each vertex deleted during a half-edge collapse operation, we compute its relative position with respect to its neighbors. Suppose that $\{\mathbf{v}_i, \mathbf{v}_j\}$ is the selected edge to be collapsed next, and $\mathbf{v}_i$ is the vertex chosen to be deleted. First, the one-ring neighbors of $\mathbf{v}_i$ before collapse are flattened over the 2D plane using a discrete conformal mapping [Lévy et al. 2002; Desbrun et al. 2002]. Then the generalized barycentric coordinates of $\mathbf{v}_i$ with respect to its one-ring neighbors [Meyer et al. 2002] are computed in the 2D plane. However, if the vertex $\mathbf{v}_i$ is on one of the matching paths (defined in Section 3.1), the one-ring of $\mathbf{v}_i$ is split into two sub-polygons, separated by the path. The barycentric coordinates of $\mathbf{v}_i$ with respect to the left and right sub-polygons are then computed: this left-side and right-side *relative location* information will be used in the coarse-to-fine map construction.



(a) (b)

Figure 3: *Mapping a hand model: (a) without background restriction, vertices in the image used as texture will move to background region in the image and cause undesirable results; (b) mapping* with *background restriction fixes the problem.*

4 Coarse-to-Fine Map Construction

Given the partial, coarse texture coordinate assignments on the base mesh and the progressive mesh built previously, we now derive texture coordinates for all the vertices of the original mesh, or at least for those we can find a proper, unambiguous assignment. The remaining ones will be dealt with in Section 5.

During this coarse-to-fine process, we reinsert vertices one at a time, in the reverse order of the previous edge collapse operations. As in [Schreiner et al. 2004], we optimize its texture coordinates by moving it around within the region formed by its one-ring neighbors in the texture domain, before optimizing each of its one-ring neighbors in the same manner. The initial texture coordinates of the inserted vertex is computed using the barycentric coordinates stored during the half-edge collapse simplification. The (u,v) texture coordinates assigned to the inserted vertex are obtained through iterative random line search [Sander et al. 2001] as we detail next.

4.1 Texture Coordinates Optimization

Based on the mesh partitioning and simplification processes, vertices can be divided into three categories. The first category consists of the feature vertices which need to be fixed in the texture domain to satisfy the constraints specified by the user. The second category includes the inner vertices of patches as well as vertices on the boundary between patches that are mapped onto the same image—that is, the vertices in this category have all of their neighboring triangles mapped into the same image. The third category includes vertices on the boundary between patches that are mapped into different images. The vertices inside regions flagged as empty are ignored for this step, and the ones on the boundary between empty and non-empty regions are treated as in the second category.

The first category of points has already been dealt with during the base mesh construction. Now, for each vertex $\mathbf{v}_i$ belonging to the second category, suppose it is reinserted (i.e., split) from $\mathbf{v}_j$. If the neighboring triangles of $\mathbf{v}_j$ are empty, then we simply do not compute texture coordinates for $\mathbf{v}_i$, and the newly introduced triangles are set as empty. Otherwise, the neighboring triangles of $\mathbf{v}_j$ are mapped into the same image, so the newly added triangles will be mapped into this image and the texture coordinates of v_i are initialized as the linear combination of texture coordinates of its one-ring neighbors in this image using the barycentric coordinates w_k previously computed during the edge collapse phase:

$$\binom{u}{v}(\mathbf{v}_i) = \sum_{\mathbf{v}_k \in \mathcal{N}(\mathbf{v}_i)} w_k \binom{u}{v}(\mathbf{v}_k), \qquad (1)$$

where $\mathcal{N}(\mathbf{v}_i)$ are the 1-ring neighbors of vertex $\mathbf{v}_i$.

In the rare occurrences where the initial texture coordinates cause triangle flipping, we place v_i at the centroid of its neighborhood polygon, as recommended in [Sander et al. 2002]. From these initial assignments, the optimization of $(u,v)(\mathbf{v}_i)$ is done using the L^∞-based geometric stretch minimization routine defined in [Sander et al. 2001]. However, we also consider the image background restriction: while guaranteeing the validity of the parameterization (no flipped triangles in texture space) by staying within the one-ring, the optimization process performs a binary search in a random direction, but rejects assignments to the background regions in the texture image. The comparison shown in Figure 3 demonstrates the necessity of the background restriction.

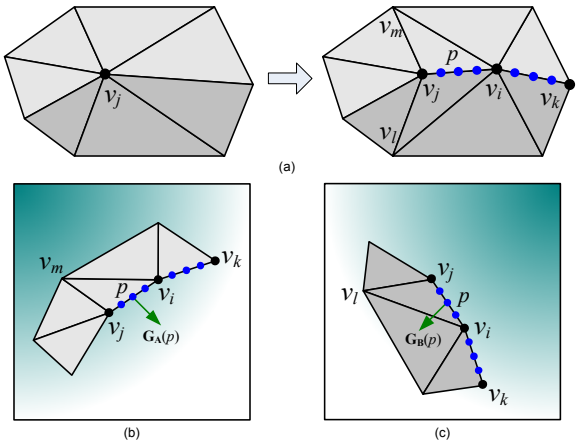


Figure 4: *Boundary vertex split and optimization: (a) a vertex $\mathbf{v}_i$ is reinserted from v_j; triangles in lighter gray are mapped onto image I_1 shown in (b) and triangles in darker gray are mapped into image I_2 shown in (c). The blue dots along $\{\mathbf{v}_i,\mathbf{v}_j\}$ and $\{\mathbf{v}_i,\mathbf{v}_k\}$ show the sampling points that are used to compute the texture mismatch energy. The green arrows in (b) and (c) indicate the image color gradients at sampling point p.*

4.2 Texture Coords Optimization Along Boundaries

Each vertex of the third category is mapped into two images. Therefore, during optimization, we need to consider not only the geometric distortion of the parameterization but also the texture *mismatch* along the patch. In this section, we describe how to initialize and optimize the texture coordinates for these particular vertices.

As shown in Fig. 4, when a vertex $\mathbf{v}_i$ belonging to a patch boundary is reinserted, it will appear in two images I_1 and I_2. The newly added triangle $\{\mathbf{v}_i,\mathbf{v}_m,\mathbf{v}_j\}$ is mapped to I_1 and $\{\mathbf{v}_i,\mathbf{v}_j,\mathbf{v}_l\}$ is mapped to I_2. The initial texture coordinates of $\mathbf{v}_i$ in I_1 $((u_1,v_1)(\mathbf{v}_i))$ can be computed by adapting Equation (1) to *only count* the neighbors mapped into I_1 (these one-sided barycentric weights were computed in Section 3.1). A similar treatment is performed for the texture coordinates $(u_2,v_2)(\mathbf{v}_i)$ in I_2.

The optimization of the vertex $\mathbf{v}_i$ in each image must now take *texture mismatch* between the two sides of the boundary into account (see Figure 5). To achieve this effect, we use a weighted energy mixing both geometric and texture constraints:

$$E_{boundary} = \lambda E_{geo} + (1-\lambda) E_{tex}, \qquad (2)$$

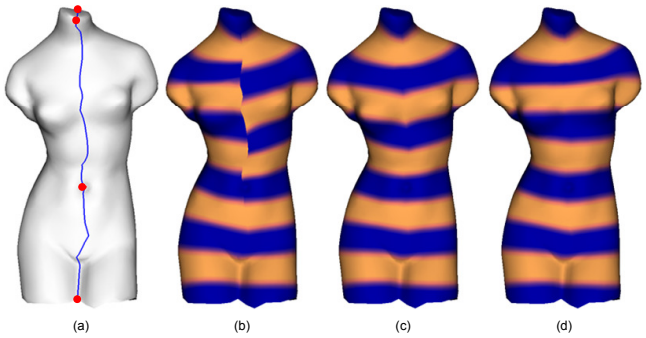


Figure 5: *An example of content-based optimization: (a) the Venus model is split right in the middle into two charts (the feature points are marked in red). (b) with geometric stretch optimization only, discontinuities appear. (c) our content-based optimization offers automatic matching of the texture colors at the boundaries. (d) C^1 smoothness can be further achieved by gradient matching.*

where E_{geo} is the L^∞-based geometric distortion measure defined in [Sander et al. 2001], and E_{tex} is a measure of the mismatch between the two texture images' content along the shared boundary as discussed in detail in the next section.

In practice, we uniformly distribute a set of points $\{s_k\}, 1 \le k \le n$, along the shared edges $\{\mathbf{v}_i, \mathbf{v}_j\}$ and $\{\mathbf{v}_i, \mathbf{v}_k\}$ as shown in Fig. 4. E_{tex} can then be computed as the sum of squared differences of the contents of image I_1 and I_2 at these sampling points:

$$E_{tex} = \sum_{k=1}^{n} \|\mathbf{I}_1(\mathbf{s}_k) - \mathbf{I}_2(\mathbf{s}_k)\|_{LUV} \tag{3}$$

(where $\mathbf{I}_k(\mathbf{s})$ indicates the color of point $\mathbf{s}$ in image I_k) for a perceptually-based LUV metric of the color space. Note that we will consider more sophisticated measures in the following section for added control.

As the sampling points may not be located at integer pixel positions, a bilinear interpolation of the image is used. The number of sampling points is decided based on the texture image resolutions, to make the distance between two successive points always less than half of the size of a pixel in both images.

To minimize the nonlinear function $E_{boundary}$, we perform random line searches [Sander et al. 2001] *alternately* on (u_1, v_1) and (u_2, v_2) in the following way:

- Fix $(u_2, v_2)(\mathbf{v}_i)$, and perform a random line search for $(u_1, v_1)(\mathbf{v}_i)$ to improve $E_{boundary}$;
- Update $\mathbf{I}_1(\mathbf{s}_i)$ and $L^\infty(t)$ for $t \in I_1$;
- Fix $(u_1, v_1)(\mathbf{v}_i)$, and perform a random line search for $(u_2, v_2)(\mathbf{v}_i)$ to improve $E_{boundary}$;
- Update $\mathbf{I}_2(\mathbf{s}_i)$ and $L^\infty(t)$ for $t \in I_2$.

The above procedure is repeated until $E_{boundary}$ cannot be further decreased; however, in all our experiments, 30 such iterations have always been enough to get satisfactory visual results. In our current implementation, E_{tex} is normalized to be within $[0, 1]$ and the weighting parameter λ is set to 0.1.

4.3 Evaluating Texture Mismatch Energy E_{tex}

Minimizing color mismatches across patch boundaries is natural for a typical texture mapping application that aims at seamlessly mapping multiple photos of a single object to a 3D model. As we just discussed, the simplest mismatch measure E_{tex} is the sum of differences of image colors. However, in applications where users want to compose features from photos of different objects over the same surface, color matching is simply not enough. More sophisticated image contents such as color gradient should be considered. For instance, one can choose a texture mismatch energy E_{tex} to be a combination of the colors and their gradients:

$$E_{tex} = \sum_{k=1}^{n} (\alpha \|\mathbf{I}_1(\mathbf{s}_k) - \mathbf{I}_2(\mathbf{s}_k)\| + (1-\alpha)\|\mathbf{G}'_1(\mathbf{s}_k) - \mathbf{G}'_2(\mathbf{s}_k)\|), \tag{4}$$

where the weighting parameter α allows users to get a proper balance between color matching and gradient matching. $\mathbf{I}_1(\mathbf{s})$ and $\mathbf{I}_2(\mathbf{s})$ are the colors of image I_1 and I_2 at $\mathbf{s}$ respectively; $\mathbf{G}'_1(\mathbf{s})$ and $\mathbf{G}'_2(\mathbf{s})$ are the color gradients transformed to the tangent space on the mesh: note that they are different from the original color gradients $\mathbf{G}_1(\mathbf{s})$ and $\mathbf{G}_2(\mathbf{s})$ of the images. We compute them in the following manner: suppose that s is located on the edge $\{\mathbf{v}_i, \mathbf{v}_j\}$ as shown in Fig. 4. We take the two surface triangles ($\{\mathbf{v}_i, \mathbf{v}_m, \mathbf{v}_j\}$ and $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_l\}$) that share the common edge $\{\mathbf{v}_i, \mathbf{v}_j\}$ and flatten them with a hinge map. The texture triangle in image I_1 corresponding to $\{\mathbf{v}_i, \mathbf{v}_m, \mathbf{v}_j\}$ and the rotated triangle define an affine mapping $\psi_{\{\mathbf{v}_i, \mathbf{v}_m, \mathbf{v}_j\}}$ from image I_1 to the plane. $\mathbf{G}'_1(\mathbf{s})$ is then computed as $\psi_{\{\mathbf{v}_i, \mathbf{v}_m, \mathbf{v}_j\}}(\mathbf{G}_1(\mathbf{p}))$. Similarly, $\mathbf{G}'_2(\mathbf{p})$ is computed as $\psi_{\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_l\}}(\mathbf{G}_2(\mathbf{p}))$. The reason for not using the original color gradient is that E_{tex} should measure the texture mismatch *on the 3D surface*, instead of in the images: the orientation and scale of images being potentially very different (see Fig. 4), computing the difference between the gradients in the images does not usually make sense.

Finally, to minimize gradient mismatch, the texture coordinates of the immediate neighbor vertices inside the boundaries of adjacent patches can be moved as well as those of boundary vertices (since they will affect the affine mapping ψ in gradient computation). Figure 5(d) shows the mapping result with gradient mismatch energy.

5 Surface Texture Inpainting

As explained previously, the constraints defined by the user may not be sufficient to assign textures on the whole surface, and texture "holes" can be present. If the user does not wish to impose more constraints with more texture images to fill in these holes, we provide a simple, yet efficient *surface texture inpainting*, where the texture colors are filled in using a Poisson-based interpolation technique.

PDE-based inpainting approaches have been shown to be quite effective for images [Bertalmio et al. 2000; Pérez et al. 2003]. Recently, [Yu et al. 2004] used the canonical Poisson equation for mesh modeling and interpolation. In this section, we propose Poisson-based editing of the *texture colors* on triangular meshes.

Setup What is known as the Poisson equation for a function f with Dirichlet boundary condition can be expressed as follows:

$$\Delta f = d \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega},$$

where Δ is the Laplace operator and d is a scalar field (for instance, the divergence of a guidance vector field $\mathbf{v}$). Ω is a closed region of an arbitrary domain (e.g., a flat image or a non-flat mesh) with boundary $\partial\Omega$. The function f^* is a known scalar function, while f is the unknown scalar function defined over Ω that can be uniquely determined by solving this Poisson equation. Since the Laplacian is a linear differential operator, the Poisson equation can be discretized into a sparse linear system that can be solved efficiently

for any discrete domain Ω. We use the cotangent formula for discrete meshes in our implementation; for details, we refer the reader to [Polthier and Preuss 2000; Pérez et al. 2003; Tong et al. 2003].

For our surface texture inpainting, we set f^* to be the color around the hole resulting from the previous texture mapping process. The function f represents the vertex colors in the hole regions we want to solve for. The Poisson equation on a mesh can only solve for colors at vertices; in order to store the whole texture on the mesh temporarily for the computation, we need to uniformly subdivide the triangles in the hole regions so that the mesh resolution matches the image resolution first. Note that we put newly inserted vertices onto the original piecewise linear mesh, so the shape of the mesh is unchanged. This subdivision allows us to solve the Poisson equation with a multigrid solver too. The refined mesh is only used for the inpainting process and the subsequent texture atlas generation: it is discarded once the inpainting process is done.

By setting d in different ways, we can get the following desirable effects:

Simple Interpolation Setting d to zero results in a harmonic interpolant; unfortunately such a color interpolation is rarely sufficient for complex textures. A better blending scheme is to derive an appropriate field d from the boundaries of the hole region. We can subdivide the immediate neighboring faces outside the boundary of the hole, and set boundary values of d as the *Laplacian of the color vector (RGB)* mapped on the refined mesh evaluated at the closest outer (refined) vertex to a boundary vertex. We then interpolate d at inner vertices using Gaussian radial basis functions (RBF), where the distance used is the usual Dijkstra's distance. Both interpolants can be computed interactively and work well for most simple, small regions in our applications. Figure 6(a) gives an example of such an interpolation.

User-guided Inpainting For hole regions surrounded by textures with salient patterns, the user can interactively specify vectors at a few vertices in the hole region to serve as blending "strokes". A non-vanishing vector field defined at every vertex is then automatically computed using RBF interpolation. After computing the boundary values of d as described in the above paragraph, the value of d at a given inside vertex is assigned as follows: by tracing the flow line forward and backward from this vertex, we find two boundary values (one at each end of the flow line); the d value is set to the weighted average of the d values at the two end vertices (where the weight is based on the distances along the flow line to the two boundary intersections). This simple procedure allows us to seamlessly stitch the texture patterns from all around the boundary as demonstrated in Figure 6(b).

Seamless Cloning By setting d directly to the Laplacian of the color values of a chosen image, we can also get a seamless *cloning tool* for surfaces. This operation requires the user to specify a source region in an arbitrary image and the associated target region on a surface [1]. We begin with the initialization of the region with a field d using a RBF-based method as described in the "Simple Interpolation" paragraph above. Then, like in PhotoMontage, we replace the values of d in the regions on the mesh covered by the user-specified image by using the Laplacian of the color mapped onto the mesh. By fixing the colors of the vertices that are *not* covered by the image region as a boundary condition, we solve the Poisson equation to get the new colors for vertices in the covered region. Our system supports preview of the cloning results by directly projecting the texture region onto the surface. Although the previewed texture is not seamlessly integrated with the surrounding surface areas, this feature still provides valuable visual feedback. Figure 6(c) shows a cloning operation on the bunny's ear.

[1] This procedure is proposed not only for non-textured holes, but also for any other mesh region as it is a convenient texturing tool.

Figure 6: *Surface texture inpainting: (a) a simple interpolation for the tail region of the feline shown in Figure 9; (b) user guided inpainting for the abdomen of the horse shown in Figure 8; (c) seamless cloning of the ear of the bunny shown in Figure 2.*

Besides filling in texture holes, Poisson equation can be further used to smooth the color discontinuity along the patch boundaries: similarly to [Agarwala et al. 2004], we can calculate the mean color between patches of each seam once textured on the object, and use Poisson equation to adjust the intensity of texture color in texture space according to the mean color. This post-processing method can efficiently suppress the residual color mismatch between different texture images.

6 Results

We have developed a texturing system based on the presented technique. Our tests and results have convinced us that *TextureMontage* offers a very flexible platform for mesh texturing. Indeed, we have been able to decorate existing 3D models by using multiple pictures (taken from the web, or photographed by ourselves) and assigning a set of feature correspondences.

The general process of specifying features proceeds as follows: first, specify the feature polygons along the contour of the images, and then specify important features that must be aligned, such as eye, nose, ankle etc. If the same feature point on surface appears in multiple images, it should be specified multiple times such that these image features are precisely matched, which is necessary because our system takes all feature points as hard constraints and their texture coordinates are fixed during optimization.

Figures 1, 7, 8 demonstrate that one can for instance take a few pictures of an animal, and apply its "fur" directly onto a 3D model. Figure 1 shows a cat model decorated by a leopard skin. Notice in this example that a leopard texture could have been generated directly using texture synthesis; however, the various anatomically-correct variations of pattern shape and color are automatically transferred when *TextureMontage* is used, offering a much more complete and versatile solution to mesh texturing. Figures 9 and 10 demonstrate the power of our system by composing textures from

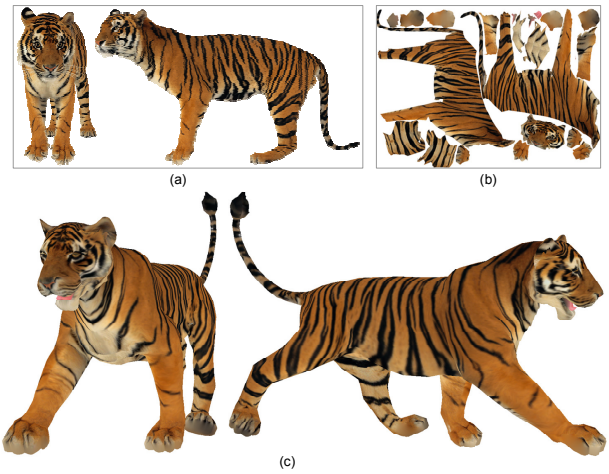


Figure 7: *Texturing a lioness model from images of a tiger: (a) input images; (b) generated texture atlas; (c) mapping results.*

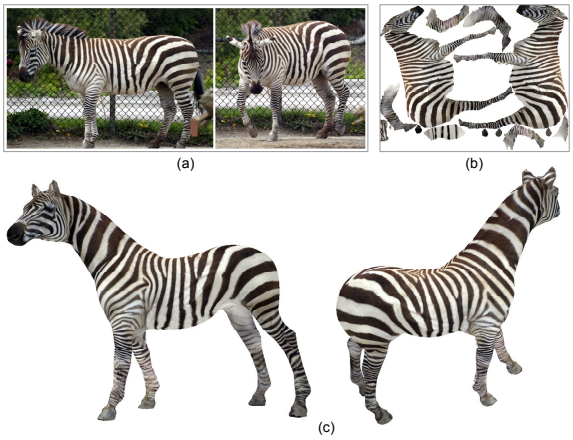


Figure 8: *Decorating a horse model using textures from a zebra: (a) input images; (b) generated texture atlas; (c) mapping results.*

multiple objects on complicated models. Specifying correspondences on the tail of the feline model (Figure 9) could have been delicate (because of the genus), but a simple interpolation fills in the hole smoothly and automatically (see Figure 6(a)). The Buddha model (Figure 10) is extremely challenging due to its abundant geometric features, high genus and also the existence of highly occluded regions like the bottom part of the gown. Texturing such a model from nine images with a reasonable amount of user interaction demonstrates the potential of our system. Automatic surface texture inpainting, especially simple interpolation, plays a crucial role in texturing a lot of hole regions. Note that a number of small charts in the texture atlas are from those hole regions.

Table 1 gives the data statistics and session time for the models presented in this paper. We selected complex 3D models and images where many features need to be carefully aligned. The timing varies from half an hour to two hours (for an untrained graduate student), depending on the number of images and feature correspondences. Specifying features after features and frequently checking the result (interactively) by computing the resulting texturing takes mostly user time. The final model requires around 1 to 3 minutes of computer time. In view of the complexity of our models and the richness of the textures, we find this amount of user interaction acceptable, and much less than using existing texturing tools that we know of.

Figure 9: *Texturing a chimeric feline model from images of multiple objects: (a) input images; (b) generated texture atlas; (c) mapping results. Here again, the use of inpainting is crucial for fast, good-looking results.*

	# vertices	# faces	# images	# features	timing (min)
cat	7,207	14,410	3	168	~ 40
bunny	15,089	30,000	6	148	~ 30
horse	20,156	40,308	2	158	~ 35
lioness	5,000	9,996	2	167	~ 45
feline	19,998	40,000	6	309	~ 90
Buddha	24,990	50,000	9	353	~ 120

Table 1: *Statistics and session time.*

7 Conclusion and Future Work

In this paper, we have demonstrated how to provide a practical mesh texturing tool that allows the user to work with a number of texture images and significantly eases the process of creating a texture atlas with nearly indiscernible patch transitions. We also offer a set of tools to the user in order to provide much freedom in the design, such as different inpainting options. There are no constraints on the number of texture images that can be used, or on the genus of the objects. If not enough constraints are given to unambiguously texture the object, we provide automatic and seamless filling.

In the future, we would like to allow the user to define some feature correspondences as *soft* constraints and optimizing their texture coordinates with respect to the image contents to further accelerate the texturing process. We are also interested in incorporating texture synthesis to our framework. The inpainting process in particular could benefit from existing automatic texture generation techniques, widening the gamut of tools available for texturing artists.

Acknowledgements

The authors would like to thank Stanford University and Robert W. Sumner for providing the 3D models used in this paper. The images of leopard, tiger and zebra were provided by Mingdong Xie. Special thanks to Kangying Cai and Jianwei Han for using our system to generate the texturing results and Steve Lin for his

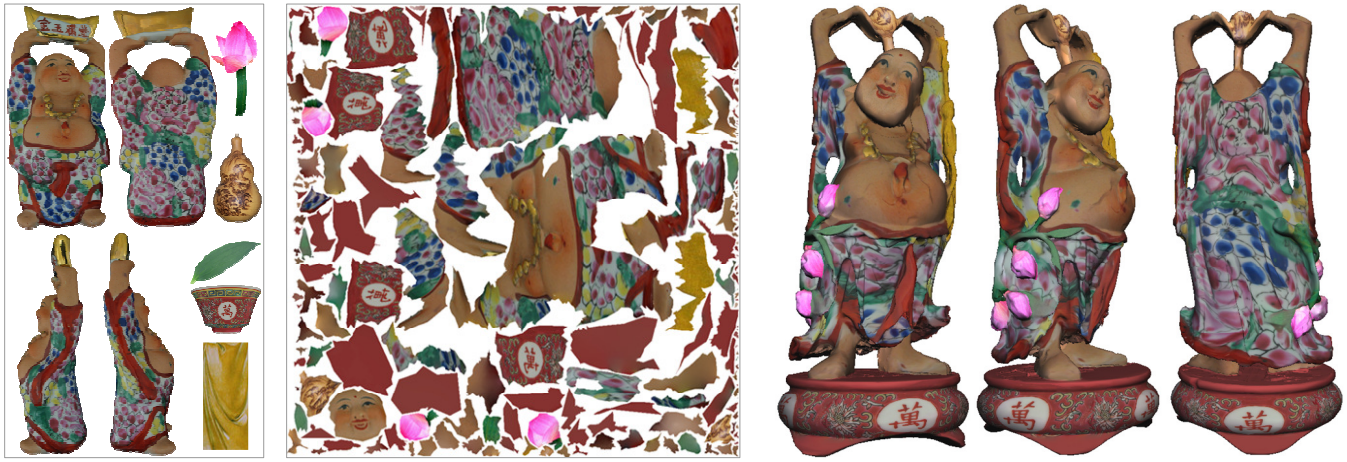


Figure 10: *Texturing Buddha from textures of multiple objects. From left to right: input images, generated texture atlas and mapping results.*

help on video production. The authors are grateful to the anonymous reviewers for their helpful suggestions and comments. The Caltech authors were partially supported by NSF (CARGO DMS-0221669 and DMS-0221666, CAREER CCR-0133983, and ITR DMS-0453145), DOE (DE-FG02-04ER25657), and Pixar.

References

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. In *Proceedings of SIGGRAPH 2004*, 294–302.

BALMELLI, L., TAUBIN, G., AND BERNARDINI, F. 2002. Space-optimized texture maps. *Computer Graphics Forum 21*, 3 (Sept), 411–420.

BERTALMIO, M., SAPIRO, G., CASELLES, V., AND BALLESTER, C. 2000. Image inpainting. In *Proceedings of SIGGRAPH 2000*, 417–424.

CARR, N. A., AND HART, J. C. 2004. Painting detail. In *Proceedings of SIGGRAPH 2004*, 842–849.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. In *Proceedings of Eurographics 2002*.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. In *Proceedings of SIGGRAPH 1995*, 173–182.

ECKSTEIN, I., SURAZHSKY, V., AND GOTSMAN, C. 2001. Texture mapping with hard constraints. *Comput. Graph. Forum 20*, 3.

FLOATER, M., AND HORMANN, K. 2003. Recent advances in surface parameterization. *Multiresolution in Geometric Modelling Workshop*.

FLOATER, M. 2003. Mean value coordinates. *CAGD 20*, 1, 19–27.

GOTSMAN, C., GU, X., AND SHEFFER, A. 2003. Fundamentals of spherical parameterization for 3d meshes. In *Proceedings of SIGGRAPH 2003*, 358–363.

GRIMM, C. M., AND HUGHES, J. F. 1995. Modeling surfaces of arbitrary topology using manifolds. *Computer Graphics 29*, Annual Conference Series, 359–368.

GU, X., AND YAU, S.-T. 2003. Global conformal surface parameterization. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 127–137.

GU, X., GORTLER, S., AND HOPPE, H. 2003. Geometry images. In *Proceedings of SIGGRAPH 2002*, 355–361.

HORMANN, K., AND GREINER, G. 1999. Mips: An efficient global parameterization method. In *Curve and Surface Design: Saint-Malo*, Vanderbilt University Press, 219–226.

IGARASHI, T., AND COSGROVE, D. 2001. Adaptive unwrapping for interactive texture painting. In *ACM Symposium on Interactive 3D Graphics*, 209–216.

KHODAKOVSKY, A., LITKE, N., AND SCHRÖDER, P. 2003. Globally smooth parameterizations with low distortion. In *Proceedings of SIGGRAPH 2003*, 350–357.

KRAEVOY, V., AND SHEFFER, A. 2004. Cross-parameterization and compatible remeshing of 3d models. In *Proceedings of SIGGRAPH 2004*, 861–869.

KRAEVOY, V., SHEFFER, A., AND GOTSMAN, C. 2003. Matchmaker: constructing constrained texture maps. In *Proceedings of SIGGRAPH 2003*, 326–333.

LEE, A., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. 1998. Maps: multi-resolution adaptive parameterization of surfaces. In *Proceedings of SIGGRAPH 1998*, 95–104.

LÉVY, 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.

LÉVY, B. 2001. Constrained texture mapping for polygonal meshes. In *Proceedings of SIGGRAPH 2001*, 417–424.

LI, Y., SUN, J., TANG, C.-K., AND SHUM, H.-Y. 2004. Lazy snapping. In *Proceedings of SIGGRAPH 2004*, 303–308.

MAILLOT, J., YAHIA, H., AND VERROUST, A. 1993. Interactive texture mapping. In *Proceedings of SIGGRAPH 1993*, 27–34.

MEYER, M., LEE, H., BARR, A., AND DESBRUN, M. 2002. Generalized barycentric coordinates on irregular polygons. *J. Graph. Tools 7*, 1, 13–22.

NEUGEBAUER, P. J., AND KLEIN, K. 1999. Texturing 3d models of real world objects from multiple unregistered photographic views. *Computer Graphics Forum 18*, 3 (Sept), 245–256.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. In *Proceedings of SIGGRAPH 2003*, 313–318.

POLTHIER, K., AND PREUSS, E. 2000. Variational approach to vector field decomposition. In *Proc. Eurographics Workshop on Scientific Visualization*.

PRAUN, E., AND HOPPE, H. 2003. Spherical parameterization and remeshing. In *Proceedings of SIGGRAPH 2003*, 340–349.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proceedings of SIGGRAPH 2000*, 465–470.

ROCCHINI, C., CIGNONI, P., MONTANI, C., AND SCOPIGNO, R. 1999. Multiple textures stitching and blending on 3d objects. In *Proceedings of the 10th Eurographics Workshop on Rendering*, Eurographics Association, 127–138.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Proceedings of SIGGRAPH 2001*, 409–416.

SANDER, P. V., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2002. Signal-specialized parametrization. In *Proceedings of the 13th Eurographics Workshop on Rendering*, Eurographics Association, 87–98.

SCHREINER, J., ASIRVATHAM, A., PRAUN, E., AND HOPPE, H. 2004. Inter-surface mapping. In *Proceedings of SIGGRAPH 2004*, 870–877.

SHEFFER, A., AND HART, J. 2002. Seamster: inconspicuous low-distortion texture seam layout. In *Proceedings of IEEE Visualization 2002*, 291–298.

SOLER, C., CANI, M.-P., AND ANGELIDIS, A. 2002. Hierarchical pattern mapping. In *Proceedings of SIGGRAPH 2002*, 673–680.

TONG, Y., LOMBEYDA, S. V., HIRANI, A. N., AND DESBRUN, M. 2003. Discrete multiscale vector field decomposition. *ACM Trans. Graphics 22*, 3, 445–452.

TURK, G. 2001. Texture synthesis on surfaces. In *Proceedings of SIGGRAPH 2001*, 347–354.

WEI, L., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of SIGGRAPH 2001*, 355–360.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. In *Proceedings of SIGGRAPH 2004*, 644–651.

ZHANG, E., MISCHAIKOW, K., AND TURK, G. 2005. Feature-based surface parameterization and texture mapping. *ACM Trans. Graphics 24*, 1, 1–27.

ZHOU, K., SNYDER, J., GUO, B., AND SHUM, H.-Y. 2004. Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 47–56.