Adobe Technical Report No. 2012-2

Non-parametric Texture Transfer Using MeshMatch

Xiaobai Chen[1], Tom Funkhouser[1], Dan B Goldman[2] and Eli Shechtman[2]
[1]Princeton University, [2]Adobe

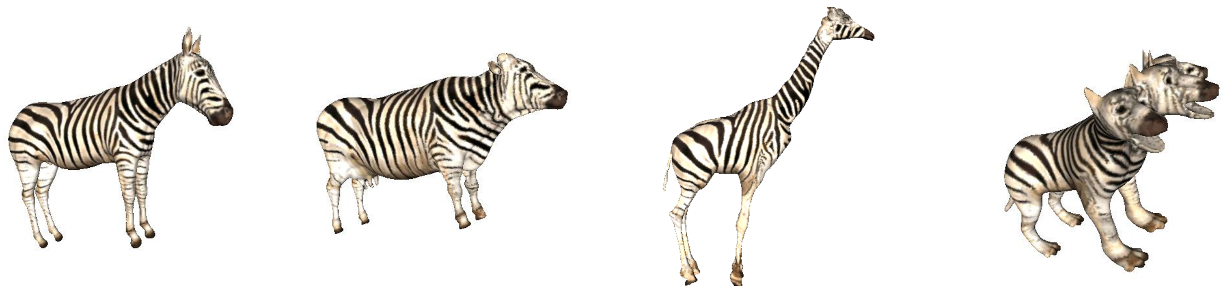

Figure 1: *Color texture transfer. From left to right: input zebra; output cow, giraffe, and cerburus (a three-headed dog).*

Abstract

Transferring surface properties such as color texture from one 3D mesh to another is a well-known problem, which has become increasingly important as the demand for textured models in games and films has grown. We propose a mesh-based analogy of the PatchMatch [Barnes et al. 2009] algorithm, which we call "MeshMatch," and a non-parametric multi-resolution approach to surface texture transfer based on this algorithm. Our system offers benefits of both parameterization and texture synthesis approaches, preserving both large- and fine-scale patterns of the source properties. We demonstrate this system in a variety of applications, including texture transfer, detail transfer, and texture painting assists.

1 Introduction

Transferring surface properties from one mesh to another is an important problem for many applications in computer graphics, including texture transfer [Mertens et al. 2006], detail synthesis [Golovinskiy et al. 2006], shape analysis [Allen et al. 2003], hole filling [Nguyen et al. 2005], and surface editing [Bhat et al. 2004].

Perhaps the most well-known instance of the problem is transferring color texture from one mesh to another. The goal is to produce a pattern on a target mesh that matches the spatially-varying pattern on a given source mesh, replicating both fine-scale textural details and large-scale structural elements of the pattern without visible distortion. This problem has become increasingly important as the demand for textured models in games and movies has exploded, while the availability of studio artists has not.

Two general approaches have been proposed to address this problem. The first aims to produce a common parameterization that can be used as a mapping from one surface to the other, usually optimizing for a smooth map that minimizes local distortions as measured by deviations in lengths and/or angles [Alexa 2002]. This strategy is able to preserve large-scale patterns in textures, but suffers from noticeable distortion in small-scale details when the source and target surfaces have significantly different shapes (e.g., two animals with different length necks). The second approach uses texture synthesis to copy patches and/or statistics of texture from the source mesh onto the target [Mertens et al. 2006]. This strategy is able to reproduce small-scale details of textures, but has difficulty preserving large-scale patterns (e.g., different width stripes on different body parts of a zebra).

In this paper, we propose a surface texture transfer framework that unifies the best features of surface parameterization and texture synthesis for 3D meshes. The heart of our framework is a surface correspondence algorithm based on PatchMatch [Barnes et al. 2009] that quickly finds the surface patch in the source with shape and color properties most similar to each patch in the target. This algorithm is largely insensitive to the dimensionality of the patch representation and thus enables association of high-dimensional feature vectors representing shape and color descriptors with each patch. Using this algorithm in a multiresolution synthesis strategy [Wei and Levoy 2001; Hertzmann et al. 2001] allows our system to efficiently transfer textures preserving both large- and fine-scale patterns of the source.

The key idea behind PatchMatch is to leverage random search and spatial coherence when finding patch correspondences. The law of large numbers suggests that some patches are likely to find the correct nearest neighbor during a random search, and coherence suggests that good correspondences can be propagated to adjacent patches, yielding full nearest neighbor fields in high-dimensional feature spaces faster than is possible with spatial indexing structures like kd-trees. However, PatchMatch was developed for images, where there is a straightforward parameterization and distances are Euclidean, and the extensions of its key steps to work for 3D surface meshes are not obvious. Non-trivial methods must be developed for selecting appropriate shape and color features, representing surface patches, searching for patch matches at arbitrary orientations, transferring features across resolutions, determining efficient propagation orders, and identifying suitable locations for random jumps. Addressing these issues is our main research contribution.

We demonstrate our surface correspondence algorithm – "MeshMatch" – in the context of a "texture-by-shape" system that uses shape as a guiding layer in the spirit of Image Analogies [Hertzmann et al. 2001]. Our approach supports several different shape features depending on the application: geodesic affinities generated from sparse manually-specified correspondences, as well as fully-automatic heat kernel signatures. We illustrate the utility of this system in a variety of applications, including texture transfer, detail transfer, and texture editing. In each case, our framework is able to automatically produce patterns on the target that resemble the source's at multiple scales, even when the source and target have relatively different shapes.

2 Related Work

Understanding how a pattern on one surface can be mapped to another is an important problem in many disciplines, including biology, paleontology, archaeology, etc. In this section, we focus on recent related work on texture transfer in computer graphics.

Parameterization. Perhaps the most obvious approach is consistent mesh parameterization. Recent work has proposed a number of ways of mapping arbitrary surfaces to canonical parameterization domains and then establishing a map with minimal distortion in those domains [Alexa 2002]. For example, consistent parameterizations have been established on spheres [Praun and Hoppe 2003; Sheffer et al. 2004], planes [Hormann et al. 2007], template meshes [Allen et al. 2003], and automatically computed base meshes [Schreiner et al. 2004]. These methods have the advantage that they can guarantee properties of the map (e.g., smoothness), but are suitable only for surfaces with similar topology and shape, as they would otherwise introduce distortions and/or seams in the transferred texture.

Mapping. Other methods compute a map between two surfaces directly via optimization with respect to an analytical distortion measure. For example, Generalized Multidimensional Scaling [Bronstein et al. 2006], Heat Kernel Maps [Ovsjanikov et al. 2010], Mobius Voting [Lipman and Funkhouser 2009], and Blended Intrinsic Maps [Kim et al. 2011] all aim to find a map minimizing deviations from isometry. Wang et al. [2008] register human faces with a conformal map and use the dense correspondences for facial expression transfer. Dinh et al. [2005] solve for a mapping between two 3D shapes using PDEs, but they assume that a smooth transformation between the shapes is already provided as input. Such methods are applicable only when the differences between two surfaces are well-approximated by an analytical model of distortion – they do not work well when different parts of the desired map require significantly different distortions (e.g., dragon $\leftrightarrow$ frog) and/or different numbers and arrangements of parts (e.g., insect with four legs $\leftrightarrow$ insect with six legs), which are typical problems of most real-world texture transfer problems.

Texture Synthesis. Other methods have transferred properties between surfaces by example-based texture synthesis. For example, Golovinsky et al. [2006] used the parametric synthesis method of Heeger and Bergen [1995] to transfer displacement maps from one face to another. Breckon and Fisher [2008] extended the non-parametric approach of Efros and Leung [1999] for 3D texture transfer and surface detail inpainting. Bhat et al. [2004] perform similar synthesis operations in the volumetric domain. These methods can be applied to arbitrary surfaces, even when the source and target have different numbers of parts. However, they have only been demonstrated for stochastic textures with highly repetetive patterns. Our work builds on the patch-based optimization approaches introduced to texture synthesis by Kwatra et al. [2005] and Wei et al. [2008] and extended to the surface domain by Han et al. [2006].

The works most closely related to ours are Mertens et al. [2006] and Lu et al. [2007]. Both papers describe "texture-by-shape" systems that use geometric surface features to guide example-based texture synthesis. They investigate methods based both on parametric and non-parametric models, finding that the parametric methods are able to reproduce weathering effects and other small-scale texture features strongly associated with local geometry. However, the non-parametric methods do not work as well, partially due to the difficulties of finding nearest neighbors in a high-dimensional search space. A goal of our work is to overcome this problem, thereby allowing synthesis of highly structured textures using non-parametric synthesis methods.

3 Overview

The objective of this work is to transfer a texture from one surface to another while retaining its geometry-aware characteristics, large-scale structural elements, and small-scale details.

To address this problem, we have developed a patch-based texture-by-shape synthesis algorithm based on Image Analogies [Hertzmann et al. 2001] and PatchMatch [Barnes et al. 2009]. As in other algorithms of this type, we simultaneously solve for the texture of the target mesh M_T and the correspondences to vertices of the source mesh M_S in a multiresolution optimization. At each level of the optimization, the texture of the target mesh is first initialized, either by up-sampling from the previous level or at random in the top level, and then an EM algorithm solves for the texture and correspondences for the next level via interleaved optimization. Specifically, the E-Step finds a correspondence from every vertex v_T of M_T to the most similar vertex v_S in the source mesh M_S, where similarity is determined by proximity of color and shape features within patches centered at v_T and v_S. The M-Step then synthesizes a new color for every vertex v_T by voting with colors found in patches of v_S corresponding to patches of M_T overlapping v_T. The iteration terminates when correspondences stabilize, or after a fixed number of iterations in each level.

A major challenge in implementing this patch-based synthesis approach is to find the source patch(es) most similar to each target patch quickly. If patch similarity is determined by the proximity of k features (e.g., $3p$ features for RGB color over a patch with p samples, plus q features for shape), then the problem is to find nearest neighbors in a k-dimensional feature space, which is well-known to be difficult when k is large, resulting in slow algorithms, even when approximate indexing structures are used [Simakov et al. 2008]. In part to avoid this problem, many previous methods for geometry-guided texture synthesis have utilized parametric synthesis algorithms, which are faster but do not replicate large-scale structural elements in synthesized textures.

The key contribution of our work is to leverage the ideas proposed in PatchMatch [Barnes et al. 2009] to accelerate the search for similar patches during non-parametric texture transfer between 3D meshes. The main idea is to leverage the law of random numbers and the spatial coherence of nearest neighbor field when searching for patch correspondences. Random guesses are interleaved with propagation to establish approximate correspondences quickly, without the need for a spatial indexing structure. As a result, finding similar patches is faster, less memory intensive, and less sensitive to the patch size and feature count. These differences enable the use of large patches with descriptive geometric features. In turn, such informative descriptors help reproduce large-scale structures in transferred textures that are consistent with the model's shape.

The following section describes the implementation of our algorithm, paying special attention to the issues unique to 3D meshes. In Section 5, we illustrate results for several possible applications, and we conclude with a brief summary and discussion of topics for future work in Section 6.

4 Algorithms

The input to our system is a source texture T_S stored on vertices of a source mesh M_S and a target mesh M_T; and, the output is a new synthesized texture T_T stored on vertices of M_T, along with an approximate nearest neighbor field NNF that associates an orientation and vertex of M_S with every vertex of M_T.

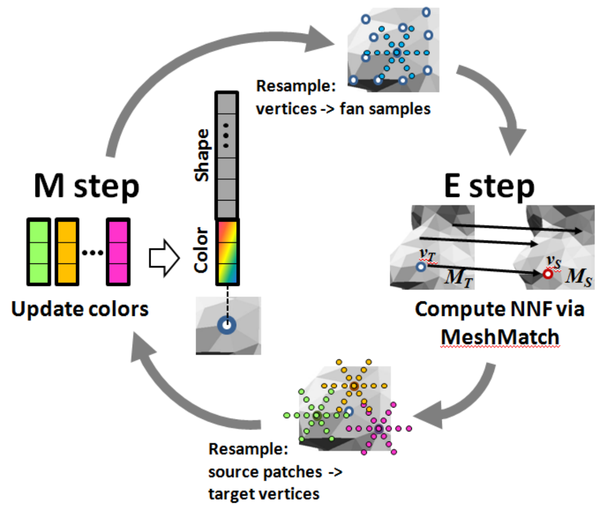

Figure 2: *Texture-by-shape algorithm overview.*

As shown in Figure 2, the system starts by preprocessing the meshes to build efficient multiresolution adjacency structures. Given a mesh M, we build L discrete levels via iterative mesh decimation using an algorithm similar in spirit to Q-Slim [Garland and Heckbert 1997]. Edges are collapsed in shortest-first order, positioning new vertices at the midpoint of the collapsed edge to preserve good aspect ratio, and retaining the "vertex tree" of parent-child relationships to allow efficient traversal between multiresolution levels. The mesh at each level, M^i, is decimated in this way until it contains approximately half as many vertices as the previous level, and the process stops when there are approximately 128 vertices in the coarsest mesh.

Then, the system performs a multiresolution optimization of the nearest neighbor field (NNF) and target texture T_T. At each multiresolution level, proceeding coarse to fine, it alternates between optimizing the NNF (via propagation and randomization) and optimizing the texture (via voting) until they jointly converge. The result is upsampled to the next multiresolution level and the process iterates. The following subsections describe each step in detail.

4.1 Patch Similarity

Before describing steps of the algorithm, we first describe the method used to determine the similarity of patches centered at vertices v_S and v_T. On the one hand, this is an implementation detail, as any distance metric could be used in the context of MeshMatch. On the other hand, it is an important detail, since the goal of the algorithm is to maximize the patch similarity $S(NNF(v_T), v_T)$ for every vertex v_T of the target mesh M_T.

There are several questions that must be addressed: 1) what is the shape of each patch?, 2) what features are stored with patches?, 3) how are the features represented in patches?, 4) how are relative orientations determined?, and 5) how is similarity computed? For images, answers to these questions are relatively straight-forward (e.g., compute L^2 differences between RGB colors in rectangular patches aligned with regularly sampled pixel locations) [Turk 2001; Wei and Levoy 2001; Kwatra et al. 2007; Xu et al. 2009]. However, they are significantly more difficult for 3D meshes, which have irregular vertex sampling and lack obvious local parameterizations.

Surface Patches. For every vertex v of every mesh at level M, we find and store the set of vertices $P(v)$ of M within a given radius r of v via a geodesic walk on vertices of the mesh. We compute a local tangent polar coordinate frame for v and store vertices of $P(v)$ in that frame. We call this representation the "patch" associated with v.

Surface Features. For each patch, we store a set of features indicating the distribution of texture values and geometric properties within the patch. Similarities of these features will be used to determine the similarity of patches.

In this work, we aim to allow both "true" geometric correspondences between semantically corresponding locations, as well as "creative" correspondences provided by an artistic user, that do not necessarily match semantic correspondences — and which also enable manual override when automatic methods fail. Our system therefore supports both automatic and interactively-specified shape features depending on the application:

Our automatic shape features must capture both small-scale details of curvature and also large-scale attributes that place the patch within a broader context of its shape (e.g., features should distinguish the neck from the back of a zebra, since different width stripes are likely to appear in those different regions). To achieve this goal, we compute the Heat Kernel Signature (HKS) at every vertex of the mesh [Sun et al. 2009]. The HKS is a smooth, orientation-invariant, and isometry-invariant shape descriptor that represents for each point p how much heat leaves p and then returns back to it in a given time t. At small time scales, it approximates the local Gauss curvature; and, at large time scales, it approximates the average diffusion distance to other points on the surface. Thus, it captures both small- and large-scale shape features in a common continuous framework. It has recently been used successfully for intrinsic symmetry detection and shape matching [Dey et al. 2010], showing excellent results for finding similarities between similar semantic regions for many types of objects. We augment the HKS vector with one more dimension: the dot product of the surface normal with a prescribed up direction. This up vector is helpful in many cases to distinguish similar shape regions with different semantics (e.g., the back and belly of an animal). Computing it is not onerous, since the up direction is prescribed in most modeling languages (e.g., positive Y in VRML), and automatic methods can be used to align the upright orientation of a mesh in most cases where it is not [Fu et al. 2008].

For interactive control, we provide an interface allowing a user to specify manual correspondences between points on different models, and compute features using the geodesic distances from each vertex to these constraint points. Unfortunately, geodesic distances alone are not very good shape features for two reasons: First, because the models have different global and local scales, the shape features do not match exactly even at correspondence points. Second, each manually-specified point has global influence across the model, making local control difficult. To address these problems we convert the geodesic distance vectors $G(v)$ to geodesic affinities $G'(v)$ using an affine transform and an exponential:

$$G'(v) = \exp(-M_{T\to S}G(v)/2\sigma^2) \tag{1}$$

The affine transform $M_{T\to S}$ (applied only to feature vectors on the target model) is computed using all pairs of correspondences (v_S, v_T) between source and target models, such that the L_2 distance $M_{T\to S}G(v_T) = G(v_S)$ exactly at all such pairs. (The resulting linear system may be under-constrained, and we find the minimum-norm solution using a linear solver.) The affine transform addresses the first problem with geodesics – meeting the constraints exactly at correspondence points – and the exponential reduces the influence of correspondences far from v. The choice of σ is not crucial: In all our results we used $\sigma = \max_{i,v} G_i(v)/3$, one third the maximum geodesic distance of any point on the mesh.

Patch Similarity. We define the dissimilarity, $D(v_T, v_S)$, of the patch centered at vertex v_T with the patch centered at v_S as the integral of the L^2 difference of all features at all points in the patch at the optimal aligning rotation:

$$D(v_S, v_T) = \text{argmin}_\theta \int_{p \in P(v_T)} \|F_T(p) - F_S(R(p, v_S, \theta))\| dp \tag{2}$$

where $P(v)$ is the surface patch centered at v, $F_T(p)$ is the feature vector associated with every surface point p, and $R(p, v_S, \theta)$ is rotation of p around v_S by θ in the tangent plane.

This definition requires a search over orientations and the evaluation of an integral for every patch comparison, which could be expensive without an appropriate patch representation. We choose it because it provides a robust dissimilarity measure and avoids the challenge of establishing a direction field on a surface.

To accelerate computation of Equation 2, we represent the distribution of features within a patch using a geodesic fan [Zelinka and Garland 2004], a representation that stores a discrete sampling of surface features in a geodesically circular structure indexed by polar coordinates (r, θ) centered at v (Figure 3). This representation provides the advantages of pre-filtering features at discrete samples for rapid comparison and rapid search over relative orientations.

In our current implementation, geodesic fans store filtered estimates of the 3 (RGB) color features at fan samples located at $R = 3$ discrete radii and $A = 18$ discrete polar angles in each fan. The shape features vary more slowly across the surface, so we find it suffices to use only a single shape vector sampled at the center of the fan (8 dimensions for HKS or anywhere from 3 to 20 dimensions for geodesic affinities). Thus we typically have up to a 182-dimensional feature vector associated with each vertex. This is not too different from typical image patch search applications, which often use 147-dimensional color patches.

4.2 Optimization

Given this definition of patch dissimilarity, our main task is to find texture values and nearest neighbors that minimize the dissimilarity measure for every vertex of the target mesh. Since the patch dissimilarity depends on both the nearest neighbor field (NNF) and the texture values (T_T), this requires a joint optimization.

Our EM optimization algorithm starts at the coarsest level of the multiresolution mesh, initializing the NNF randomly. Then, for each multiresolution mesh level, from coarse to fine, it updates the NNF via propagation and randomization in the E-Step, and then

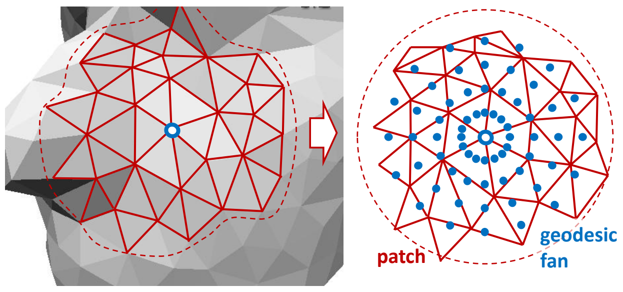

Figure 3: *Data structures. Left: 3D mesh (gray) and local patch region (red); Right: patch mapped into local 2D coordinate frame (red) with geodesic fan samples (blue).*

reconstructs an estimate of the target texture from the NNF with a voting algorithm in the M-Step. After the EM iterations have converged, the NNF is upsampled to the next finer resolution, the target texture is reconstructed at the finer level by re-voting, and the process is repeated until the finest multiresolution level is reached.

NNF Propagation. During the NNF propagation phase of the E-Step, our goal is to utilize "good" correspondences found in the previous step (possibly by random chance) to find others for adjacent vertices. To do this, we visit each vertex v_T of M_T in order, updating NNF(v_T) to be the vertex v_S of M_S with highest patch similarity to v_t amongst a set of candidates that includes its current nearest neighbor, NNF(v_T), and all vertices adjacent to nearest neighbors of vertices adjacent to v_T. This phase takes advantage of spatial correspondence in the nearest neighbor field to propagate good correspondences from one vertex to others in its connected component of the mesh.

An issue encountered for 3D meshes in this phase is how to choose a vertex traversal order. For maximal efficiency, we aim to find an ordering that favors visiting adjacent vertices in succession, so that recent updates can be leveraged as each vertex is visited. For images, the natural order is to visit pixels across successive scan-lines. However, no such order exists for vertices of a mesh. So, we select a start vertex randomly within each connected component of M_T and visit vertices in breadth-first order from there. In successive iterations we start from the last vertex in the most recent sequence; this is analogous to alternating forwards/backwards passes in the image-based PatchMatch algorithm.

Note that some care must be taken to propagate consistently under patch rotations. As described in Barnes et al. [2010], when searching over rotations, propagation neighborhoods in the target domain must be transformed to the appropriate coordinate frame.

NNF Randomization. During the NNF randomization phase of the E-Step, our goal is to "explore" the space of possible nearest neighbors. To do this, we visit each vertex v_T of M_T, updating NNF(v_T) to be the vertex with highest patch similarity to v_t amongst a set of candidates that includes its current nearest neighbor, NNF($_T$), and vertices sampled from M_S at increasing distances from NNF(v_T). Following PatchMatch, the candidate set includes a random sample within radius r of v_T, another within radius $2r$, and so on. This random sample often contains a good correspondence for v_T, which is propagated to its adjacent vertices in the next iteration.

An interesting issue encountered for meshes in this phase is how to sample vertices matching a desired distribution of distances from a given vertex NNF(v_T). This is easy for images, since distances are Euclidean. However, for meshes, distances are geodesic, and tracing geodesic paths for long distances on a mesh is too slow for the inner loop of our algorithm. To address this issue, we leverage the hierarchy of the multiresolution mesh. Since every multiresolution level covers the same area with half as many vertices as the next, they are spaced approximately $\sqrt{2}$ times as far apart. Thus, we can randomly sample vertices at prescribed distances of $\sqrt{2}^j$ from NNF(v_T) by traversing j levels up the multiresolution hierarchy from NNF(v_T) and then traversing back down j levels moving to a random vertex in the next finer level at each step down (Figure 4). This provides a fast way to sample random vertices from uniform distributions over several region sizes.

Texture Reconstruction: The M-Step of the optimization updates the target texture based on the current estimate of the nearest neighbor field. For every vertex v_T of M_T, the algorithm performs a weighted averaging (voting) of values from the source texture T_S using the correspondences of nearby vertices (Figure 5). Specifically, for every vertex v_p in the neighborhood of v_T, the loca-

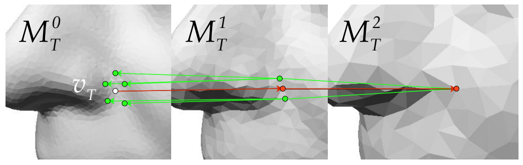

Figure 4: *Using the multiresolution mesh hierarchy to sample random vertices at multiple distances from v_T (shown in white): Ancestor vertices (shown in red) are associated with many descendant vertices (shown in green), allowing for fast random sampling of a large region by hopping multiple levels in the multi-resolution mesh.*

tion of v_T is mapped to a location on M_S by NNF(v_p), and then a color estimate is accumulated from the source mesh vertices using a weighted average, with weights falling off as a Gaussian with the distance from v_T to produce the final reconstruction value.

This Gaussian filtering is essential in the inner loop of the algorithm to avoid aliasing, as otherwise feedback loops can amplify spurious patterns. However, the resulting output is somewhat blurrier than the input texture, so as an optional last step, we reconstruct per-vertex colors at the finest resolution by simply copying the nearest-neighbor color instead of filtering with a weighted Gaussian. The results in Figure 1 and Figure 7 were produced using this nearest-neighbor sampling.

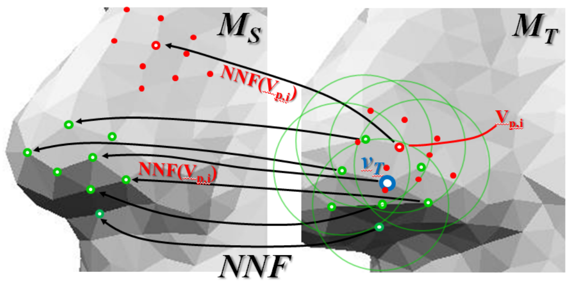

Figure 5: *Texture Reconstruction. Geodesic fan samples from the NNF vote for the texture values of each vertex v_T.*

Upsampling. Finally, after the EM optimization has converged for level i, we upsample the NNF to the next finer level $i + 1$. If NNF(v_T^i) = v_S^i at level i, we must determine the corresponding vertex v_S^{i+1} and relative coordinate frame for v_T^{i+1} in the next finer level of the multiresolution hierarchy. To do this, we project each vertex v_T^{i+1} onto M_T^i and determine the geodesic offset and relative orientation with respect to v_T^i. Then, we use NNF(v_T^i) to find v_S^i and reverse the geodesic offset and relative orientation to find the mapped position of vertex v_T^{i+1} on M_S^i. Finally, we map that position and orientation to the finer mesh M_S^{i+1} and find the closest vertex v_S^{i+1}. This provides the upsampled value: NNF(v_T^{i+1}) $= v_S^{i+1}$.

5 Results and applications

In this section, we investigate the performance and applications of the proposed algorithms.

5.1 User interface

For applications requiring manual correspondence points, we provide a user interface allowing the user to add and remove corresponding points. It supports creation and deletion of both one-to-one and one-to-many constraints, and allows camera motions of source and target views to be locked, facilitating navigation around the models while identifying corresponding points.

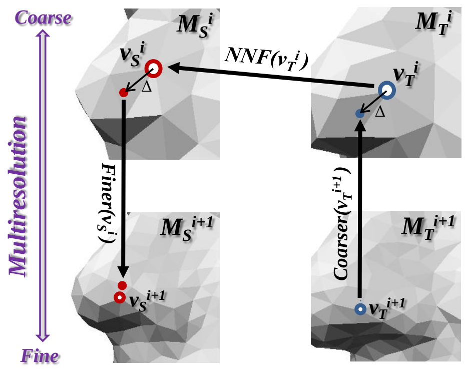

Figure 6: *Upsampling. At the end of each iteration, NNFs for vertices in multiresolution level i are upsampled from the NNF of level $i + 1$.*

In addition, the computation of the texture transfer is fast enough that we can provide progressive updates of the transfer in progress in the target view, enabling the user to decide whether additional correspondences are required to improve the results. Coarse-resolution results are typically visible within a few seconds after adding new correspondences, and fine resolutions are computed in under 30 seconds. Please refer to our supplementary video to see this interface in use.

5.2 Color texture transfer

The most straightforward application of our approach is the transfer of color texture from one 3D model to another. Figure 1 illustrates transfer of textures from a zebra to three other creatures. Note that when transferring the zebra's stripes to the other geometry, the resulting stripes are not uniformly or randomly distributed as one might expect using texture synthesis approaches. Instead they retain the same spatial variation as seen in the source model, with widely separated stripes near the hindquarters and thicker, more densely packed stripes along the neck. However, the texture correspondence is not one-to-one: The giraffe's neck has many more stripes than the zebra, and the cerberus has three noses and six eyes. This highlights an advantage of our method: Because it does not assume one-to-one mapping between source and destination geometry, it can adapt appropriately to variations in local shape, synthesizing more texture in stretched areas. Thus, our method demonstrates the preservation of high level structure like direct mapping techniques, while maintaining the flexibility of non-parametric texture synthesis approaches.

Even in cases where a one-to-one mapping is possible, it may be extremely deformed and thus unsuitable for direct texture mapping. In Figure 9 we compare our method to correspondences computed using blended intrinsic maps[Kim et al. 2011]. These correspondences – and indeed any one-to-one mapping for this model pair – produce severe distortions in order to match such dramatically different geometry. Our method can use any one-to-one or one-to-many mapping method as initialization and/or soft constraint, synthesizing locally consistent and distortion-free texture while retaining global semantic correspondences.

For color transfer we find that models such as the cow and zebra typically require only about 6 to 10 manual correspondences, and the weights of shape and color features are set to be equal (after normalizing for dimensionality and variance). In some cases we use lower shape weighting at finer scales to encourage more flexible transfer.

Figure 7 illustrates transfer between more widely differing models. Here there are few geometric cues because there is no obvious correspondence between shapes. By adjusting a few correspondence points we can obtain a global correspondence and synthesize a new spatially-varying texture that is locally similar to the original at those points.

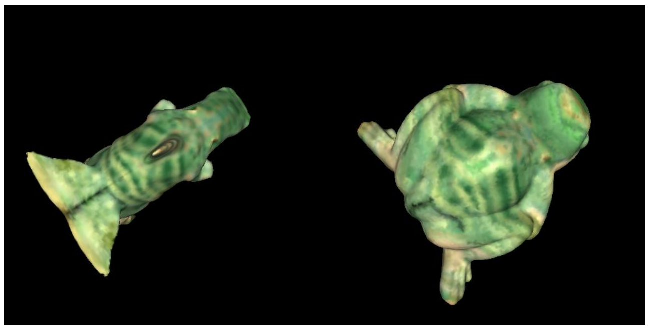

Figure 7: *Transfer between widely differing models. Left: input mesh and texture. Middle: output texture with correspondences clicked at head, tail, tip of leg and center of back and belly.*

5.3 Texture painting power assists

Texture painting in film and game production can require a great deal of manual effort. What's worse, that effort can be for naught if the topology or geometry of a model is modified after painting, requiring reparameterization and repainting due to distortion. We propose two uses of MeshMatch to reduce effort in the texture painting workflow: Texture by example and texture update.

Texture by example: In addition to transfer between models, MeshMatch can also be used to transfer texture within a model. This can be useful, for example, to propagate painted textures automatically from regions already painted to regions of a model that have not yet been touched. Figure 8 illustrates some examples. Although textures were manually created for just one surface region, they are successfully propagated across the rest of the surface. In this case, the object is globally self-similar, so we use automatically-computed HKS features rather than requiring manual correspondence. However, note that matching HKS features alone (Figure 8c) cannot properly resolve the differently colored stripes – only by using local texture patches can we synthesize the proper complex color pattern seen in the source.

Mertens et al. [2006] also use shape features to guide texture synthesis. However, their method uses smaller texture patches and mostly local shape features. In Figure 8d we demonstrate the impact of patch size and shape feature dimensionality by reducing the fan radii to $R = 2$ and using only one dimension of fine-scale HKS. Because of the efficiency of the MeshMatch matching algorithm, we are able to integrate shape and color features into a single optimization over a high dimensional space. Although Mertens et al. use a broader combination of several different features, their method handles only stochastic textures with a simple statistical correlation to the underlying shape features, whereas our method can handle essentially arbitrary relationships by using nearest-neighbor search.

Texture update: Figure 10 depicts a creature that has been significantly lengthened after textures were painted. In this case we resynthesized the texture map for the entire torso and copied textures verbatim for other body parts. However, if the geometric modeling package retains knowledge about the portions of geometry that were edited and those that were not, we could make finer grained updates by synthesizing new texture only in modified regions. No explicit reparameterization is required.

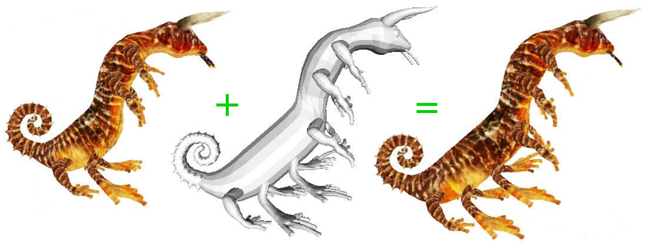

Figure 10: *Reapplying texture after geometry changes. Left to right: input geometry and texture, modified geometry, texture updated to fit new geometry.*

5.4 Facial detail transfer

Another possible application is transferring fine-scale surface details from one surface to another. For example, Figure 11 shows how pores, wrinkles, and creases captured in a highly detailed scan of a face can be used to synthesize plausible details on smooth face scan. To do this, we separate the detailed face scan into a smooth component and a displacement map, and then use the shape features to guide transfer of the displacement map onto the target smooth face. Notice that many details have been synthesized plausibly, including folds in the eyelids, both crows' feet around the eyes, and the nasal labial folds extending downwards from the nostrils along the cheeks. These results compare favorably with Golovinsky et al.[2006], who provided the input data for this example – their parametric synthesis algorithm does not replicate such large scale structures.

5.5 Performance

Our system runs at interactive rates. The MeshMatch portion of the algorithm takes only 5 iterations and 6.5 seconds to converge for a mesh with 64K vertices, each represented by a 182-dimensional feature vector. This requires no extra storage for spatial indexing structures, beyond the nearest neighbor field itself.

The texture transfer system based on MeshMatch employs several mesh processing steps that are not fast, but can be performed offline. Specifically, for a mesh with 64K vertices, building the multiresolution data structure takes 120 seconds and building geodesic fans takes 40 seconds. However, the remainder of the pipeline runs at near-interactive rates: at the finest scale (64K vertex mesh), finding nearest neighbors with MeshMatch takes 6.5 seconds per iteration, and reconstructing the texture from the correspondence field takes 5.5 seconds per iteration. These runtimes scale linearly with the number of vertices, so a rough preview at 16K vertices takes only about 3 seconds. Using five EM iterations at five multiresolution levels, the total texture transfer time is approximately 100 seconds. Please see our accompanying video for a real-time demonstration of our system in use.

Several further speed improvements are possible: For example, MeshMatch presently uses a fast randomized search for vertices but exhaustive search over orientations. Barnes et al. [Barnes et al. 2010] have demonstrated that image patch search over orientations

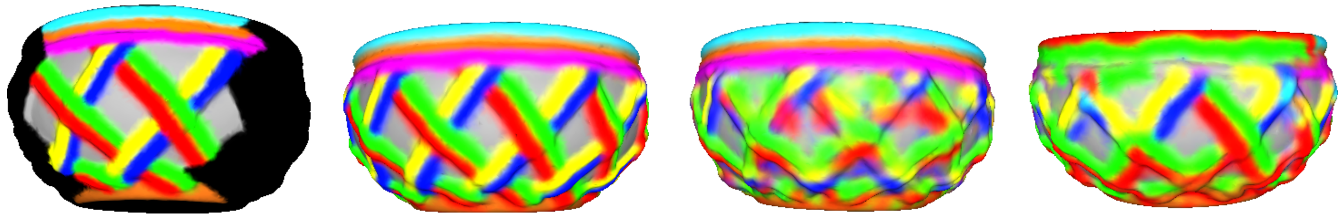

Figure 8: *Texture painting by example. Left to right: (a)input meshes with partial painted texture, (b)output meshes with texture propagated to unpainted regions, (c) output meshes by running our algorithm only with shape (d) output meshes by using a smaller fan with only small scale HKS to mimic Mertens et al. [2006]. Notice that (b-d) all show the opposite side of the bowl from the side shown on (a).*

can also be performed efficiently using randomized search with propagation, so we believe augmenting our method using a similar technique could result in a possible additional 10x speedup.

6 Conclusion and Future Work

This paper has described a surface correspondence algorithm based on PatchMatch [Barnes et al. 2009] and demonstrated its use in a variety of texture transfer applications. The most important advantage of this algorithm is that it enables finding nearest neighbors between surface points in a high-dimensional feature space encoding large-scale geometric and color features. As a result, nonparameteric texture synthesis algorithms can be used to replicate large-scale patterns like stripes from a zebra and creases from a face, in addition to small scale patterns like pores on a face.

Our current implementation is just a first prototype and has several limitations that suggest directions for future work.

First, our system has some parameters that must be adjusted for each example. The main parameters we have adjusted are the starting scale and the different shape-vs.-color weights for each level of the multiresolution hierarchy – these are adjusted based on the size of patterns in the source texture. While these parameters can be useful for artistic control, it would be interesting to investigate future methods to set them automatically based on scale-space analysis of the source data.

Second, our interactive shape features were designed for efficient computation, and not quality. In particular, the mapping becomes difficult to control when more than about 15 correspondences are added, as the influence of each point affects a large portion of the model. In addition, one-to-many correspondences create extreme distortion in the shape mapping in regions where the one-to-one mapping is split to one-to-many. Although our method can still synthesize plausible texture in these regions, better methods for both global and local control are desirable.

Finally, it will be interesting to study which applications can best leverage the types of surface correspondences found by MeshMatch. Unlike much of the previous work in inter-surface mapping, our algorithm does not aim to find a smooth, bijective map between surfaces with low distortion everywhere. Rather, it aims to find correspondences between points with similar features. We have demonstrated that correspondences of this type are useful for texture and label transfer in computer graphics, but it will be interesting to see what other types of applications can benefit from this approach (e.g., computer vision, archaeology, art, etc.).

Acknowledgements

We thank NSF (CCF-0937139, CNS-0831374) for partial funding of this project.

References

ALEXA, M. 2002. Recent advances in mesh morphing. *Computer Graphics Forum 21*, 2, 173–198.

ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2003. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graph. (SIGGRAPH) 22*, 3, 587–594.

BARNES, C., SHECHTMAN, E., FINKELSTEIN, A., AND GOLDMAN, D. B. 2009. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH) 28*, 3 (Aug.).

BARNES, C., SHECHTMAN, E., GOLDMAN, D. B., AND FINKELSTEIN, A. 2010. The generalized PatchMatch correspondence algorithm. In *European Conference on Computer Vision*.

BHAT, P., INGRAM, S., AND TURK, G. 2004. Geometric texture synthesis by example. In *Proc. Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 41–44.

BRECKON, T. P., AND FISHER, R. B. 2008. 3D surface relief completion via non-parametic techniques. *IEEE Trans. Pattern Analysis and Machine Intelligence 30*, 12 (Dec.), 2249 – 2255.

BRONSTEIN, A. M., BRONSTEIN, M. M., AND KIMMEL, R. 2006. Efficient computation of isometry-invariant distances between surfaces. *SIAM J. Scientific Computing*.

DEY, T. K., LI, K., LUO, C., RANJAN, P., SAFA, I., AND WANG, Y. 2010. Persistent heat signature for pose-oblivious matching of incomplete models. *Computer Graphics Forum (Symposium on Geometry Processing)*.

DINH, H., YEZZI, A., AND TURK, G. 2005. Texture transfer during shape transformation. *ACM Trans. Graphics 24*, 2 (April).

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *IEEE Int'l. Conf. on Comp. Vis.*

FU, H., COHEN-OR, D., DROR, G., AND SHEFFER, A. 2008. Upright orientation of man-made objects. *ACM Trans. Graph. 27*, 3.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH 1997*, Comp. Graph. Proc., Annual Conf. Series, 209–216.

GOLOVINSKIY, A., MATUSIK, W., PFISTER, H., RUSINKIEWICZ, S., AND FUNKHOUSER, T. 2006. A statistical model for synthesis of detailed facial geometry. *ACM Trans. Graph. (Proc. SIGGRAPH) 25*, 3.

HAN, J., ZHOU, K., WEI, L., GONG, M., BAO, H., ZHANG, X., AND GUO, B. 2006. Fast example-based surface texture synthesis via discrete optimization. *The Visual Computer 22*, 9, 918–925.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. In *SIGGRAPH '95: Proc. 22nd annual conf. Comp. graph. and interactive techniques*, 229–238.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *SIGGRAPH '01: Proc. 28th annual conf. Comp. graph. and interactive techniques*, 327–340.

HORMANN, K., LÉVY, B., AND SHEFFER, A. 2007. Mesh parameterization: Theory and practice. In *SIGGRAPH 2007 Course Notes*, ACM Press, San Diego, CA, no. 2, vi+115.

KIM, V., LIPMAN, Y., AND FUNKHOUSER, T. 2011. Blended intrinsic maps. *ACM Transactions on Graphics (TOG) 30*, 4, 79.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Transactions on Graphics, SIGGRAPH 2005* (August).

KWATRA, V., ADALSTEINSSON, D., KIM, T., KWATRA, N., CARLSON, M., AND LIN, M. 2007. Texturing fluids. *IEEE Trans. Visualization and Computer Graphics 13*, 939–952.

LIPMAN, Y., AND FUNKHOUSER, T. 2009. Mobius voting for surface correspondence. *ACM Transactions on Graphics (Proc. SIGGRAPH) 28*, 3 (Aug.).

LU, J., GEORGHIADES, A. S., GLASER, A., WU, H., WEI, L.-Y., GUO, B., DORSEY, J., AND RUSHMEIER, H. 2007. Context-aware textures. *ACM Transactions on Graphics 26*, 1.

MERTENS, T., KAUTZ, J., CHEN, J., AND DURAND, F. 2006. Texture transfer using geometry correlation. In *Eurographics Symposium on Rendering*.

NGUYEN, M. X., YUAN, X., AND CHEN, B. 2005. Geometry completion and detail generation by texture synthesis. *The Visual Computer (Special Issue for Pacific Graphics 2005) 21*, 9–10, 669–678.

OVSJANIKOV, M., MÉRIGOT, Q., MÉMOLI, F., AND GUIBAS, L. 2010. One point isometric matching with the heat kernel. In *Eurographics Symposium on Geometry Processing (SGP)*.

PRAUN, E., AND HOPPE, H. 2003. Spherical parametrization and remeshing. *ACM Trans. Graph. 22*, 3, 340–349.

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

SHEFFER, A., GOTSMAN, C., AND DYN, N. 2004. Robust spherical parameterization of triangular meshes. *Computing 72*, 1-2, 185–193.

SIMAKOV, D., CASPI, Y., SHECHTMAN, E., AND IRANI, M. 2008. Summarizing visual data using bidirectional similarity. In *CVPR*.

SUN, J., OVSJANIKOV, M., AND GUIBAS, L. J. 2009. A concise and provably informative multi-scale signature based on heat diffusion. *Comput. Graph. Forum 28*, 5, 1383–1392.

TURK, G. 2001. Texture synthesis on surfaces. In *SIGGRAPH '01: Proc. 28th annual conf. Comp. graph. and interactive techniques*, 347–354.

WANG, S., GU, X., AND QIN, H. 2008. Automatic non-rigid registration of 3d dynamic data for facial expression synthesis and transfer. In *CVPR 2008*, 1–8.

WEI, L.-Y., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH '01: Proc. 28th annual conf. on Comp. graph. and interactive techniques*, 355–360.

WEI, L., HAN, J., ZHOU, K., BAO, H., GUO, B., AND SHUM, H. 2008. Inverse texture synthesis. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, 52.

XU, K., COHEN-OR, D., JU, T., LIU, L., ZHANG, H., ZHOU, S., AND XIONG, Y. 2009. Feature-aligned shape texturing. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia 2009) 28*, 5.

ZELINKA, S., AND GARLAND, M. 2004. Similarity-based surface modelling using geodesic fans. In *SGP '04: Proc. 2004 Eurographics/ACM SIGGRAPH Symp. on Geom. Proc.*, 204–213.

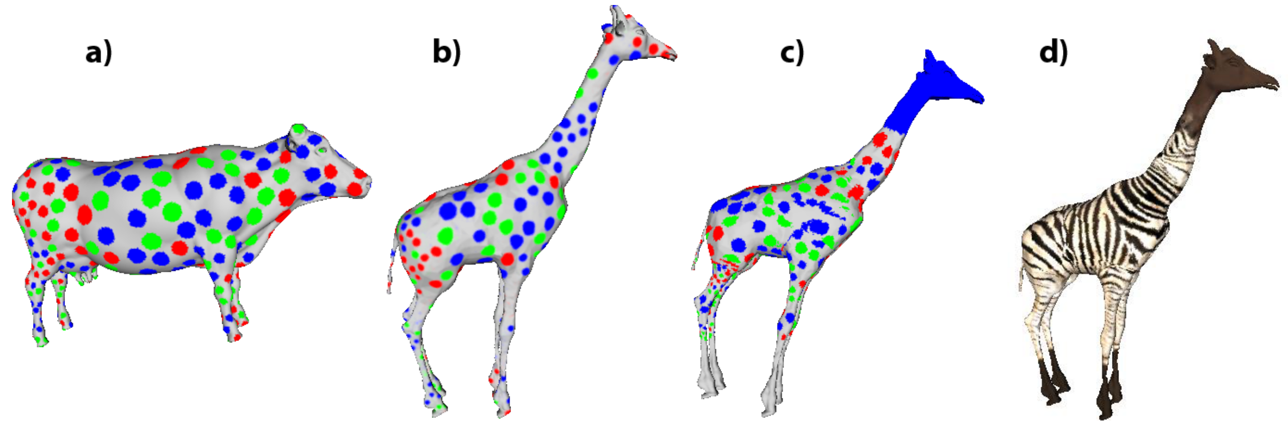

Figure 9: *Left to right: (a) input cow with polkadot, transfered results of (b) ours, (c) of using Blended Intrinsic Maps (BIM), and (d) zebra stripes transfered also using BIM.*

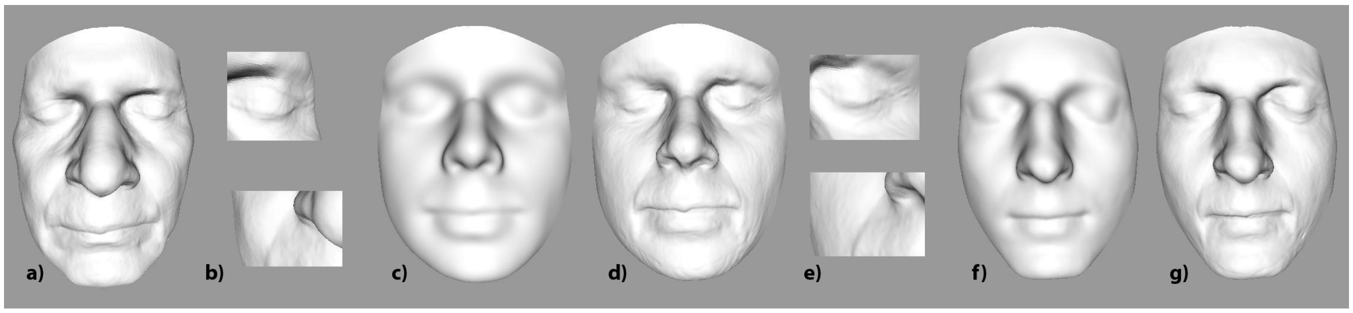

Figure 11: *Face detail transfer. (a) SRC detailed face (b) DST smooth face 1 with detail transfered (c) DST smooth face 2 with detail transfered.*