Appearance-Space Texture Synthesis

Sylvain Lefebvre Hugues Hoppe
Microsoft Research

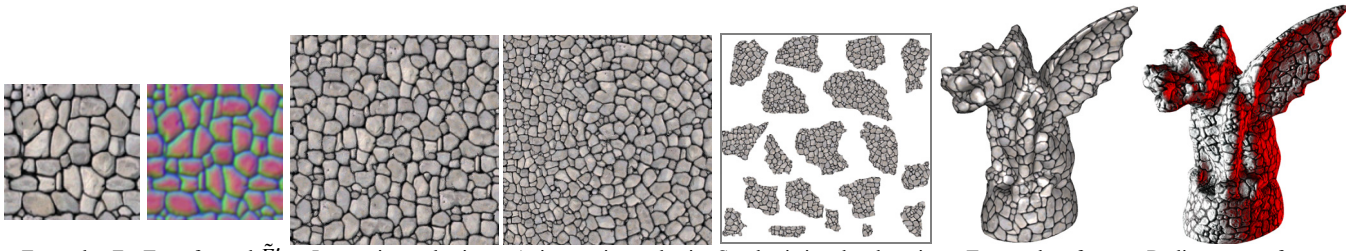


Figure 1: Transforming an exemplar into an 8D appearance space $\tilde{E}'$ improves synthesis quality and enables new real-time functionalities.

Abstract

The traditional approach in texture synthesis is to compare color neighborhoods with those of an exemplar. We show that quality is greatly improved if pointwise colors are replaced by appearance vectors that incorporate nonlocal information such as feature and radiance-transfer data. We perform dimensionality reduction on these vectors prior to synthesis, to create a new appearance-space exemplar. Unlike a texton space, our appearance space is low-dimensional and Euclidean. Synthesis in this information-rich space lets us reduce runtime neighborhood vectors from 5×5 grids to just 4 locations. Building on this unifying framework, we introduce novel techniques for coherent anisometric synthesis, surface texture synthesis directly in an ordinary atlas, and texture advection. Remarkably, we achieve all these functionalities in real-time, or 3 to 4 orders of magnitude faster than prior work.

Keywords: exemplar-based synthesis, surface textures, feature-based synthesis, anisometric synthesis, dimensionality reduction, RTT synthesis.

1. Introduction

We describe a new framework for exemplar-based texture synthesis (Figure 1). Our main idea is to transform an exemplar image E from the traditional space of pixel colors to a space of appearance vectors, and then perform synthesis in this transformed space (Figure 2). Specifically, we compute a high-dimensional appearance vector at each pixel to form an *appearance-space* image E', and map E' onto a low-dimensional *transformed exemplar* $\tilde{E}'$ using principal component analysis (PCA) or nonlinear dimensionality reduction. Using $\tilde{E}'$ as the exemplar, we synthesize an image S of exemplar coordinates. Finally, we return $E[S]$ which accesses the original exemplar, rather than $\tilde{E}'[S]$.

The idea of exemplar transformation is simple, but has broad implications. As we shall see, it improves synthesis quality and enables new functionalities while maintaining fast performance.

Several prior synthesis schemes use appearance vectors. Heeger and Bergen [1995], De Bonet [1997], and Portilla and Simoncelli [2000] evaluate steerable filters on image pyramids. Malik et al [1999] use multiscale Gaussian derivative filters, and apply clustering to form discrete *textons*. Tong et al [2002] and Magda and Kriegman [2003] synthesize texture by examining inter-

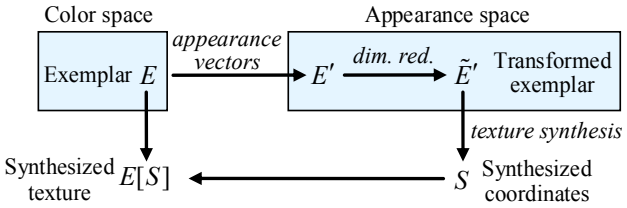


Figure 2: Overview of synthesis using exemplar transformation.

texton distances. However, textons have two drawbacks: the clustering introduces discretization errors, and the distance metric requires costly access to a large inner-product matrix. In contrast, our approach defines an appearance space that is continuous, low-dimensional, and has a trivial Euclidean metric.

The appearance vector at an image pixel should capture the local structure of the texture, so that each pixel of the transformed exemplar $\tilde{E}'$ provides an information-rich encoding for effective synthesis (Section 3). We form the appearance vector using:

- *Neighborhood* information, to encode not just pointwise attributes but local spatial patterns including gradients.
- *Feature* information, to faithfully recover structural texture elements not captured by local L^2 error.
- *Radiance transfer*, to synthesize material with consistent mesoscale self-shadowing properties.

Because exemplar transformation is a preprocess, incorporating the neighborhood, feature, and radiance-transfer information has little cost. Moreover, the dimensionality reduction encodes all the information concisely using *exemplar-adapted* basis functions, rather than generic steerable filters.

In addition we present the following contributions:

- We show that exemplar transformation permits parallel pixel-based synthesis using a runtime neighborhood vector of just 4 spatial points (Section 4), whereas prior schemes require at least 5×5 neighborhoods (and often larger for complex textures).
- We design a scheme for high-quality anisometric synthesis. The key idea is to maintain texture coherence by only accessing immediate pixel neighbors, and to transform their synthesized coordinates according to a desired Jacobian field (Section 5).
- We create surface texture by performing anisometric synthesis directly in the parametric domain of an ordinary texture atlas. Because our synthesis algorithm accesses only immediate pixel neighbors, we can jump across atlas charts using an indirection map to form seamless texture. Prior state-of-the-art schemes

[e.g. Sloan et al 2003; Zhang et al 2003] require expensive per-vertex synthesis on irregular meshes with millions of vertices, and subsequently resample these signals into a texture atlas. Our technique is more elegant and practical, as it operates completely in the image space of the atlas domain, never marching over a mesh during synthesis (Section 6).

- Finally, we describe an efficient scheme for advecting the texture over a given flow field while maintaining temporal coherence (Section 7). Our results exhibit less blurring than related work by [Kwatra et al 2005].

Previous work in these various areas required minutes of computation time for a static synthesis result. Remarkably, appearance-space synthesis lets us perform *all* the above functionalities *together* in tens of milliseconds on a GPU, i.e.

feature-preserving synthesis and advection of consistent radiance-transfer texture anisometrically mapped onto an arbitrary atlas-parameterized surface, in real-time.

Because we can synthesize the texture from scratch every frame, the user may interactively adjust all synthesis parameters, including randomness controls, direction fields, and feature scaling. Moreover, by computing the Jacobian map on the GPU, even the surface geometry itself can be deformed without any CPU load.

2. Background on texture synthesis

Our pixel-based neighborhood-matching synthesis scheme builds on a long sequence of earlier papers, which we can only briefly review here. The traditional approach is to generate texture sequentially in scanline order, comparing partially synthesized neighborhoods with exemplar neighborhoods to identify the best matching pixel [Garber 1981; Efros and Leung 1999]. Improvements include hierarchical synthesis [Popat and Picard 1993; De Bonet 1997; Wei and Levoy 2000], coherent synthesis [Ashikhmin 2001], precomputed similarity sets [Tong et al 2002], and order-independent synthesis [Wei and Levoy 2003].

We extend the parallel approach of [Lefebvre and Hoppe 2005], in which synthesis is realized as a sequence of GPU rasterization passes, namely upsampling, jitter, and correction. All passes operate on an image pyramid S of exemplar coordinates rather than directly on exemplar colors (Figure 3). The key step of interest to us is the correction pass, in which each $S[p]$ is assigned the exemplar coordinate u whose 5×5 neighborhood $N_E(u)$ best matches the currently synthesized neighborhood $N_S(p)$.

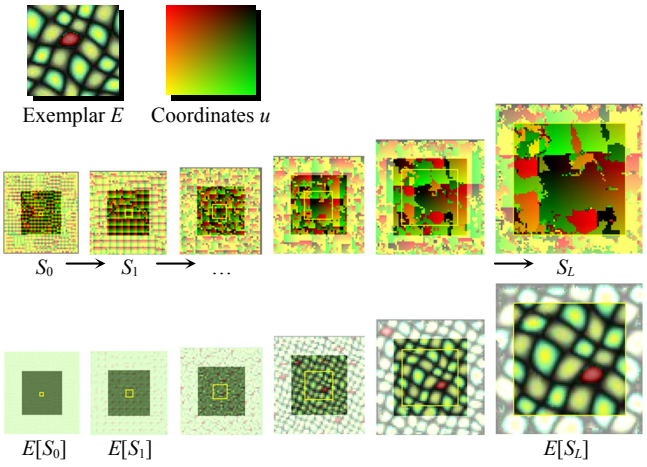


Figure 3: Review of parallel texture synthesis.

3. Definition of appearance vector

3.1 Spatial neighborhood

To compare a synthesized neighborhood $N_S(p)$ and exemplar neighborhood $N_E(u)$, distance is typically measured by summing squared color differences. Because each pixel only contributes information at one point, large neighborhoods are often necessary to accurately recreate the original texture structure. Such large neighborhoods are a runtime bottleneck, as they require both many memory references and an expensive search process.

The runtime search can be accelerated by recognizing that the set of image neighborhoods typically lies near a lower-dimensional subspace. One technique is to project neighborhoods using PCA [Hertzmann et al 2001; Liang et al 2001; Lefebvre and Hoppe 2005]. The runtime-projected $\tilde{N}_S = PN_S$ is compared against the precomputed $\tilde{N}_E = PN_E$. However, note the apparent inefficiency of the overall process – a large vector N_S must be gathered from memory and multiplied by a large matrix P, to then only yield a low-dimensional vector $\tilde{N}_S$.

Our insight is to apply neighborhood projection on the exemplar itself as a precomputation, and then perform synthesis using this transformed exemplar. While we still perform PCA to accelerate runtime neighborhood matching (Section 4), our contribution is to redefine the signal contained in the neighborhood itself!

More concretely, let the Gaussian-weighted 5×5 neighborhoods of an RGB exemplar E define a 75D appearance-space exemplar E'. We then project the exemplar using PCA to obtain a 3D transformed exemplar $\tilde{E}'$. Note in Figure 4 how $\tilde{E}'$ has a greater "information density" than E. The figure also demonstrates that synthesis using $\tilde{E}'$ has higher quality than using E even though both have 3 channels and hence the same synthesis cost. (Here we use the synthesis scheme described later in Section 4)

Generally, we let the transformed exemplar $\tilde{E}'$ be 8D rather than 3D to further improve synthesis quality (Figure 4). The additional spatial bases are especially useful to encode feature and radiance-transfer data as introduced in the next sections. Note that for many color textures, a 4D transformed exemplar is sufficient, as shown in the supplemental material and in Figure 14.

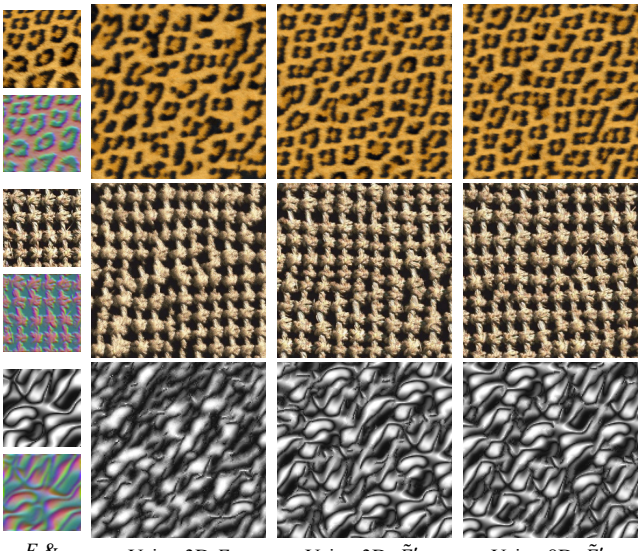


Figure 4: Benefit of using exemplar transformation with spatial neighborhood as appearance vector.

3.2 Feature distance

Small spatial neighborhoods cannot encode large texture features. More importantly, simple color differences often fail to recognize semantic structure (e.g. mortar between nonhomogeneous stones). Wu and Yu [2004] introduce the notion of a feature mask to help guide the synthesis process. Their patch-based scheme applies local warping to align texture edges. We next show how their idea can be easily incorporated within our pixel-based scheme.

Given a user-provided binary feature mask, we compute a signed-distance field, and include this distance as an additional image channel prior to the neighborhood analysis of Section 3.1. Thus the new appearance vector has 5×5×4=100 dimensions, but is still projected using PCA into 8D. For some textures, we find it beneficial to apply a simple remapping function to the distance. For example, clamping the distance magnitude to some maximum helps suppress singularities along the feature medial axis.

Note that unlike [Wu and Yu 2004], we need not consider feature tangent direction explicitly, because it is derived automatically in the spatial neighborhood analysis. (The PCA transformation may even detect feature curvature.) Moreover, "tangent consistency" is also captured within the appearance-space Euclidean metric. In fact, we obtain preservation of texture features *without any change whatsoever to the runtime synthesis algorithm*.

Figure 5 compares synthesis results before and after inclusion of feature distance in the appearance vector. The weight w given to the feature-distance channel can be varied as shown in Figure 6. The tradeoff is that a larger weight w downplays color differences, eventually resulting in synthesis noise.

Another scheme with the same goal of feature preservation is the two-pass approach of [Zhang et al 2003], which first synthesizes a binary texton mask by matching large hierarchical neighborhoods (with $15^2+11^2+7^2+3^2=305$ samples), and then uses this binary mask as a prior for color synthesis. In comparison, our approach involves a single synthesis pass with much smaller neighborhood comparisons (4 samples), and runs 4 orders of magnitude faster.

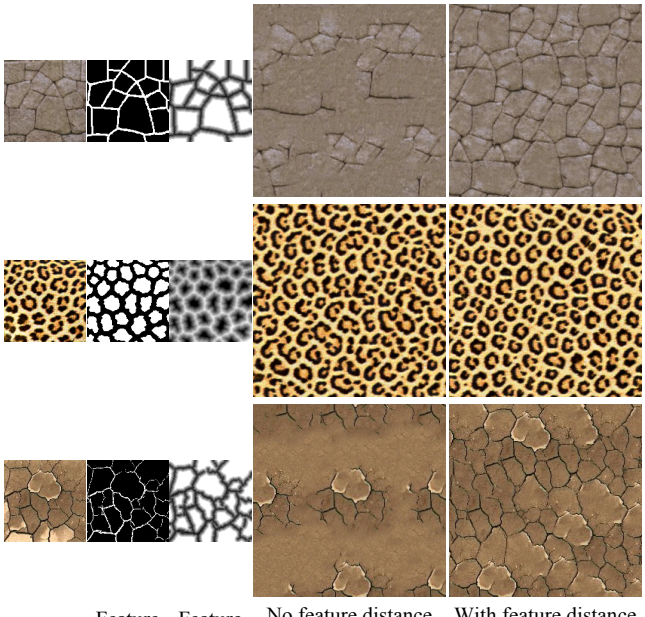


Figure 5: Inclusion of feature signed-distance in the appearance vector, to better preserve semantic texture structures.

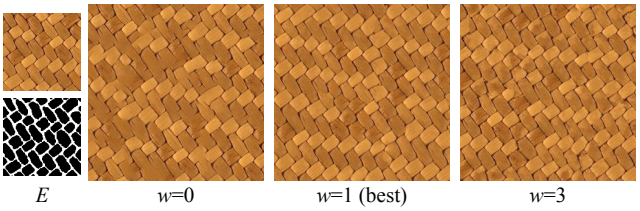


Figure 6: Effect of feature channel weight on synthesis quality.

3.3 Radiance transfer

Realistic rendering of complex materials requires not only point-wise attributes but also mesoscale effects like self-shadowing and parallax occlusion. Tong et al [2002] synthesize a bidirectional texture function (BTF) to capture surface appearance under all view and light directions. They cluster the high-dim. reflectance vectors onto a discrete set of 3D textons [Leung and Malik 2001]. BTFs represent reflectance using directional bases for both view and light, and are therefore ideal for point light sources.

We chose to represent a radiance transfer texture (RTT) which instead uses spherical harmonics bases appropriate for low-frequency lighting environments [Sloan et al 2003]. To simplify our system, we implement the diffuse special case which omits view-dependence but still retains self-shadowing. The RTT is computed from a given patch of exemplar geometry using ray tracing [Sloan et al 2003]. For accurate shadows, we use spherical harmonics of degree 6, so each RTT pixel is 36-dimensional.

We redefine the appearance vector as a 5×5 neighborhood of the RTT texture, i.e. a vector of dimension $5^2 \cdot 36=900$. As before these are PCA-projected into an 8D appearance-space exemplar. For efficient PCA computation, we skip the covariance matrix by instead using iterative expectation maximization [Roweis 1997]. Again, the runtime synthesis algorithm is unchanged.

Even though the 8D transformed exemplar loses ~30-50% of the appearance-space variance (Section 9), the mesoscale texture structure is sufficiently well captured to allow accurate RTT synthesis. As can be seen in Figure 7 and in the video, we obtain consistent self-shadowing under a changing lighting environment.

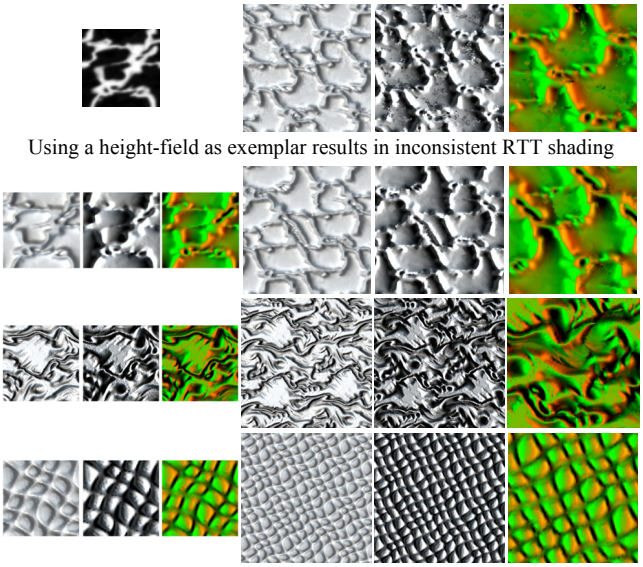
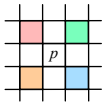


Figure 7: Diffuse radiance transfer as appearance vector, to obtain consistent self-shadowing during RTT shading.

4. Isometric synthesis

Having created an 8D appearance-space exemplar, we can apply any pixel-based synthesis algorithm, e.g. evaluating a 5×5 neighborhood error by summing squared differences (in 8D rather than 3D). But in fact, the greater information density permits synthesis using a more compact runtime neighborhood.

In adapting our earlier parallel synthesis algorithm [Lefebvre and Hoppe 2005], we find that a runtime neighborhood of just 4 diagonal points is sufficient:

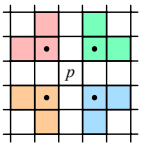


, i.e. $N_S(p) = \left\{ \tilde{E}'\left[S[p+\Delta]\right] \;\middle|\; \Delta = \begin{pmatrix}\pm 1\\ \pm 1\end{pmatrix} \right\}$.

However, the parallel synthesis correction algorithm operates as a sequence of subpasses, and all 4 diagonal points belong to the same subpass, resulting in poor convergence. To improve convergence without increasing the size of the neighborhood vector $N(p)$, we use the following observation. For any pixel p', a nearby synthesized pixel $p'+\Delta'$ can predict the synthesized coordinate at p' as $S[p'+\Delta']-\Delta'$. Thus, for each point $p+\Delta$, we average together the predicted appearance vectors from 3 synthesized pixels used in different subpasses. Specifically, we use the combination

$$N_S(p;\Delta) = \tfrac{1}{3}\sum_{\Delta'=M\Delta,\, M\in\mathcal{M}} \tilde{E}'\left[S\left[p+\Delta+\Delta'\right]-\Delta'\right]$$

where $\mathcal{M} = \left\{ \begin{pmatrix}0&0\\0&0\end{pmatrix}, \begin{pmatrix}1&0\\0&0\end{pmatrix}, \begin{pmatrix}0&0\\0&1\end{pmatrix} \right\}$ accesses the neighboring pixels shown inset. Although we now read a total of 12 pixels, the neighborhood vector $N_S(p)$ still only has dimension 4·8=32. For the anisometric synthesis scheme described in the next section, we find it useful to re-express the neighborhood using the equivalent formula



$$N_S(p;\Delta) = \tfrac{1}{3}\sum_{\Delta''=\Delta+M\Delta,\, M\in\mathcal{M}} \tilde{E}'\left[S\left[p+\Delta''\right]-\Delta''+\Delta\right].$$

Then, we compare the synthesized neighborhood vector $N_S(p)$ with precomputed vectors $N_E(u)$ in the exemplar to find the best-matching exemplar pixel:

$$S[p] := \operatorname{argmin}_{u\in\mathcal{C}(p)} \left\| N_S(p) - N_E(u) \right\|.$$

As in [Tong et al 2002], we limit the search to a set of k-coherent candidates

$$\mathcal{C}(p) = \left\{ C_i\left(S[p+\Delta]\right) - \Delta \;\middle|\; i=1\ldots k,\ \|\Delta\| < 2 \right\},$$

where the precomputed similarity set $\{C_{1\ldots k}(u)\}$ identifies other pixels with neighborhoods similar to that of u. (We use k=2.)

As in [Lefebvre and Hoppe 2005], we speed up runtime neighborhood comparisons by applying PCA projection (not to be confused with the PCA used in exemplar transformation). Specifically, we project the 32D exemplar neighborhoods to 8D as $\tilde{N}_E = P' N_E$ where P' is a 8×32 matrix. And, we use the same projection $\tilde{N}_S = P' N_S$ at runtime, so that evaluating the distance $\left\|\tilde{N}_S(p) - \tilde{N}_E(u)\right\|^2$ requires just three GPU instructions.

To summarize, the preprocess performs two successive PCA projections, $E' \to \tilde{E}'$ and $N(\tilde{E}') \to \tilde{N}(\tilde{E}')$. All our results derive from this basic scheme.

Synthesis quality is greatly improved over [Lefebvre and Hoppe 2005] as can be seen in Figure 1 and in our supplemental material, available at http://research.microsoft.com/projects/AppTexSyn/.

5. Anisometric synthesis

In this section we generalize synthesis to allow local rotation and scaling of the texture according to a Jacobian field J. Rather than defining multiple versions of the exemplar texture under different deformations [Taponecco and Alexa 2004], we anisometrically warp the synthesized neighborhood N_S prior to neighborhood matching, as in Ying et al [2001]. One advantage is the ability to reproduce arbitrary affine deformations, including shears and nonuniform scales. In our setting, the method of Ying et al would define the warped synthesized neighborhood as

$$N_S(p;\Delta) = \tilde{E}'\left[S\left[p+\varphi(\Delta)\right]\right], \text{ with } \varphi(\Delta) = J^{-1}(p)\Delta,$$

where differences from the isometric scheme are colored blue. That is, the sampling pattern in synthesis space is transformed by the inverse Jacobian at the current point. However, such a transformation requires filtered resampling since the samples no longer lie on the original grid. More significantly, if the Jacobian has stretch (i.e. spectral norm greater than unity), the warped samples become discontiguous, resulting in a breakdown in texture coherence. Ying et al [2001] also describe a coherent scheme that warps neighborhoods in exemplar space, but this inhibits search acceleration techniques such as our neighborhood PCA $\tilde{N}$.

Instead, we seek to estimate an anisometrically warped neighborhood vector $N_S(p)$ by only accessing immediate neighbors of p. Our idea is to use the *direction* of each offset vectors $\varphi(\Delta)$ to infer which neighboring pixel to access, and then to use the full offset vector $\varphi(\Delta)$ to transform the neighbor's synthesized coordinate.

More precisely, we gather the appearance vector $N_S(p;\Delta)$ for each neighbor Δ as follows. We normalize the Jacobian-transformed offset as $\delta = \bar{\varphi}(\Delta) = \left\lfloor \varphi(\Delta)/\|\varphi(\Delta)\| + 0.5 \right\rfloor$, which keeps its rotation but removes any scaling. Thus $p+\delta$ always references one of the 8 immediate neighbors of pixel p. We retrieve the synthesized coordinate $S[p+\delta]$, and use it to predict the synthesized coordinate at p as $S[p+\delta]-J(p)\delta$, much as in Section 4 but adjusting for anisometry. Finally, we offset this predicted synthesized coordinate by the original exemplar-space neighbor vector Δ. As before, we compute the appearance vector as a combination of 3 pixels. The final formula is

$$N_S(p;\Delta) = \tfrac{1}{3}\sum_{\delta''=\bar{\varphi}(\Delta)+\bar{\varphi}(M\Delta),\, M\in\mathcal{M}} \tilde{E}\left[S\left[p+\delta''\right]-J(p)\delta''+\Delta\right].$$

Also, we redefine the k-coherent candidate set as

$$\mathcal{C}(p) = \left\{ S[p+\Delta] + C'_i\left(S[p+\Delta]\right) - J(p)\Delta \;\middle|\; i=1\ldots k,\ \|\Delta\| < 2 \right\}$$

to account for anisometry. Because the Jacobian-transformed offsets introduce continuous deformations, the synthesized coordinates $S[p]$ are no longer quantized to pixel locations of the exemplar. Therefore, to preserve this fine-scale positioning of synthesized coordinates, we re-express the precomputed similarity sets as offset vectors rather than absolute positions. Because the synthesized coordinates are continuous values, exemplar accesses like $\tilde{E}[u]$ involve bilinear interpolation, but this interpolation is inexpensive in the hardware texture sampler.

Finally, we maintain texture coherence during coarse-to-fine synthesis by modifying the upsampling pass to account for the anisometry. Each child pixel inherits the parent synthesized coordinate, offset by the Jacobian times the relative child location:

$$S_l[p] := S_{l-1}[p-\Delta] + J(p)\Delta, \quad \Delta = \left(\pm 1/2 \;\; \pm 1/2\right)^T.$$

Figure 8 shows some example results. Our accompanying video shows interactive drawing of texture orientation and scaling, which is an exciting new tool for artists.

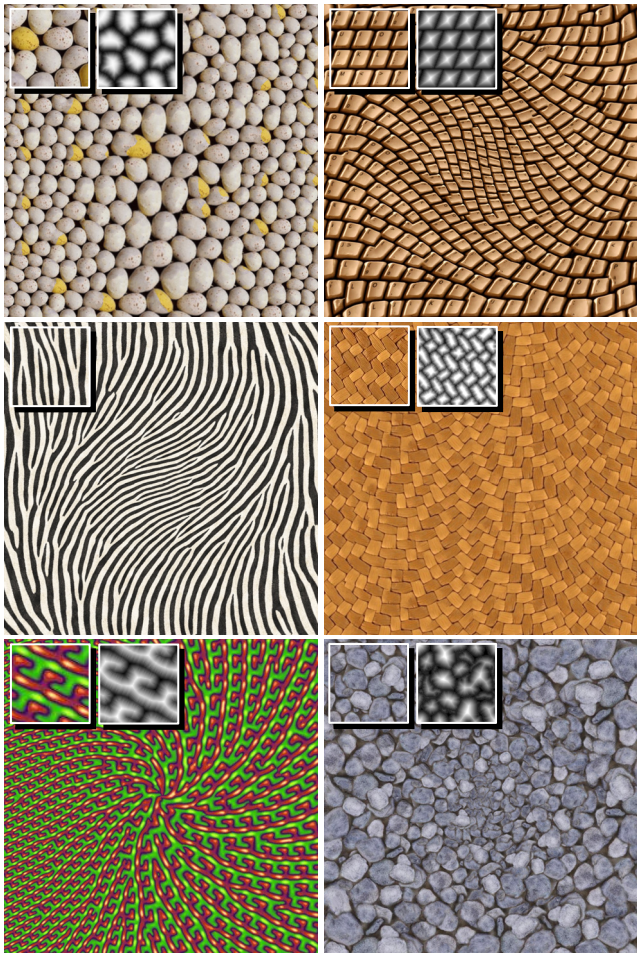


Figure 8: Results of anisometric synthesis.

6. Surface texture synthesis

Anisometric synthesis is important for creating surface texture. Approaches include per-vertex methods [e.g. Turk 2001; Wei and Levoy 2001] and patch-based ones [e.g. Neyret and Cani 1999; Praun et al 2000; Magda and Kriegman 2003]. To allow efficient parallel evaluation, we directly synthesize pixels in the parametric domain of the surface, like Ying et al [2001]. But whereas they construct overlapping charts on a subdivision surface, we consider ordinary texture atlases on arbitrary triangle meshes.

Surface tangential field. The user specifies a surface field t,b of tangent and binormal vectors (Figure 9). This field can be interpolated from a few user constraints [Praun et al 2000] or obtained with a global optimization [Hertzmann and Zorin 2000].

Anisometry. Our goal is to synthesize texture anisometrically in the parametric domain such that the surface vectors t,b are locally identified with the standard axes $\hat{u}_x,\hat{u}_y$ of the exemplar. From Figure 9 we see that $(t\ b) = J_f J^{-1} I$, where J_f is the 3×2 Jacobian of the surface parameterization $f : D \to M$, and J is the desired 2×2 Jacobian for the synthesized map $S : D \to E$. Thus,

$$J = (t\ b)^+ J_f = \left((t\ b)^T (t\ b)\right)^{-1} (t\ b)^T J_f ,$$

where "+" denotes matrix pseudoinverse. If $(t\ b)$ is orthonormal, then $(t\ b)^+ = (t\ b)^T$. The parameterization is piecewise linear, so the Jacobian J_f is piecewise constant within each triangle. In contrast, the tangential frame $(t\ b)$ varies per-pixel.

We compute the Jacobian map J on the GPU by rasterizing the surface mesh over its texture domain. The pixel shader evaluates $J_f = (\mathrm{ddx}(f)\ \mathrm{ddy}(f))$ using derivative instructions, which is exact since J_f is constant during the rasterization of each triangle.

Indirection map. To form seamless texture over a discontinuous atlas, the synthesis neighborhoods for pixels near chart boundaries must include samples from other charts. Here we exploit the property that our anisometric correction scheme accesses a neighborhood of fixed extent. We read samples across charts using a per-level indirection map I, by replacing each access $S[p]$ with $S[I[p]]$. These indirection maps depend only the surface parameterization, and are precomputed by marching across chart boundaries. We reserve space for the necessary 2-pixel band of indirection pointers around each chart during atlas construction. Because all resolution levels use the same atlas parameterization, extra gutter space is reserved at the finest level (Figure 10). We avoid running the correction shader on the inter-chart gutter pixels by creating a depth mask and using early z culling.

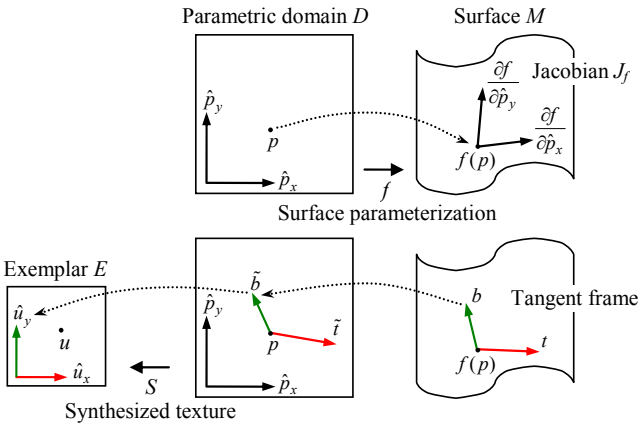


Figure 9: For surfaces, the synthesis Jacobian involves both the surface parameterization and a specified surface tangential field.

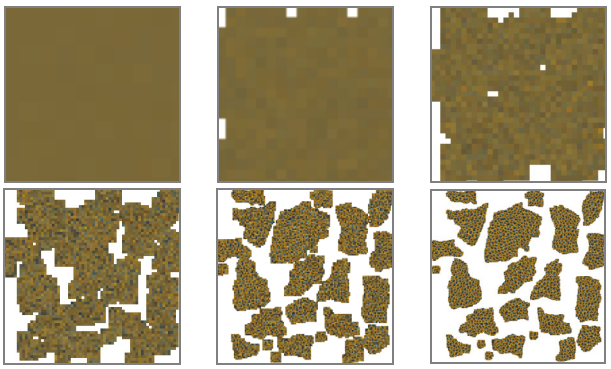
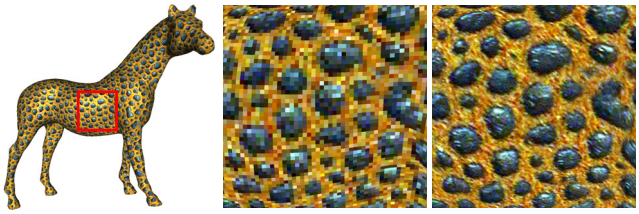


Figure 10: Levels 1-6 of the multiresolution synthesis pyramid.



Textured surface | No magnif.; 12.3 fps | With magnif.; 11.7 fps

Figure 11: Surface texture synthesis with magnification.

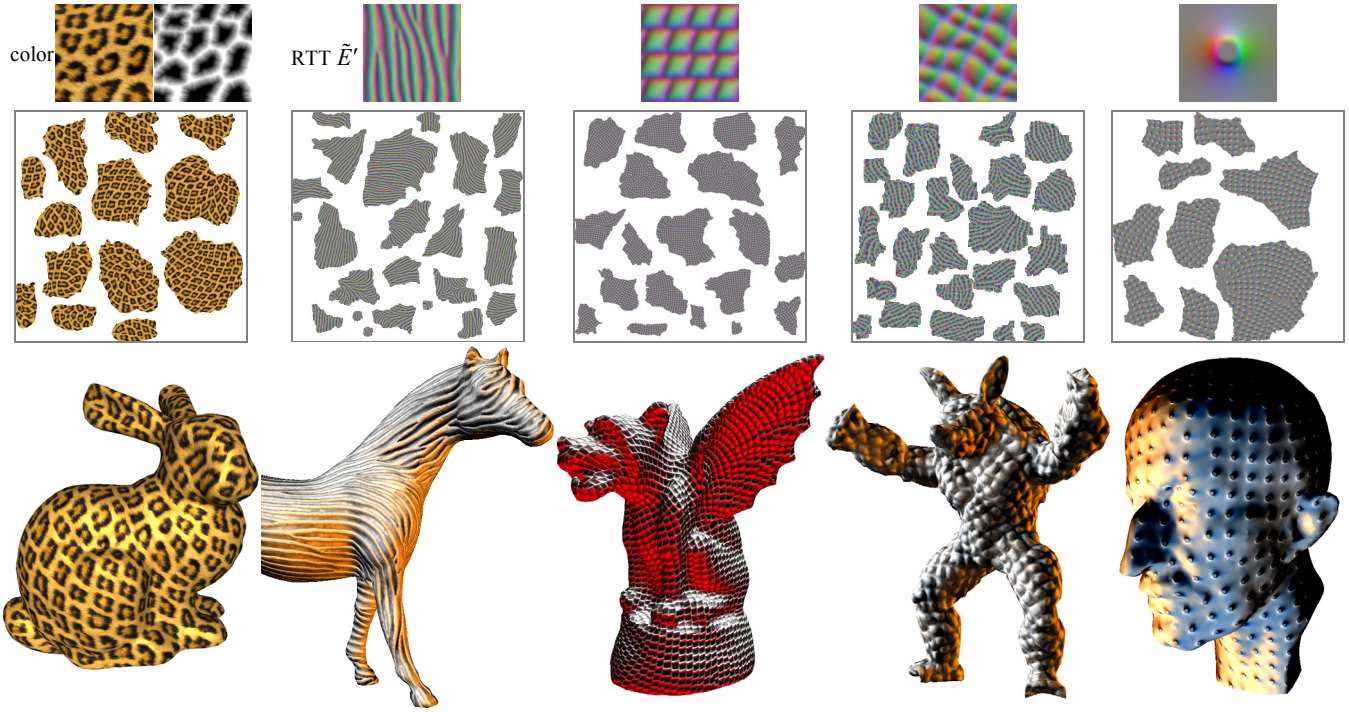


Figure 12: Results of surface texture synthesis. The first column is an example of color texture, while the next four columns show radiance-transfer textures. As in other figures, we visualize only the first 3 channels of the 8D transformed exemplar $\tilde{E}'$.

Anisometric synthesis magnification. One difficulty in synthesizing texture within an atlas is that some parameterization distortion is usually inevitable and leads to undersampled regions. We are able to hide the sampling nonuniformity using synthesis magnification [Lefebvre and Hoppe 2005]. The idea is to use the synthesized coordinates S to access a higher-resolution exemplar E_H. Specifically, the pixel value at a continuous coordinate p is obtained by combining the 4 nearest synthesized pixels as

$$\mathrm{Mag}_{E_H}(p) = \sum_{\Delta=p-\lfloor p\rfloor-\delta,\ \delta\in\left\{\binom{0}{0},\binom{1}{0},\binom{0}{1},\binom{1}{1}\right\}} w(\Delta)\, E_H\left[S[p-\Delta]+\Delta\right],$$

where $w(\Delta) = |\Delta_x| \cdot |\Delta_y|$ are bilinear interpolation weights. We modify synthesis magnification to account for anisometry by accessing the Jacobian map:

$$\mathrm{Mag}_{E_H}(p) = \sum_{\Delta=p-\lfloor p\rfloor-\delta,\ \delta\in\ldots} w(\Delta)\, E_H\left[S[p-\Delta]+J(p-\Delta)\Delta\right].$$

Anisometric synthesis magnification is performed in the surface shader at rendering time and thus adds little cost (Figure 11). Additional results are presented in Figure 12, including four examples of radiance-transfer textures (discussed in Section 3.3).

7. Texture advection

Texture can be synthesized in space-time with a nonzero velocity field. Applications include texture-based flow visualization and textured animated fluids (e.g. water, foam, or lava). The challenge is to maintain spatial and temporal continuity without introducing blurring or ghosting. Neyret [2003] blends several advecting layers of texture regenerated periodically out-of-phase, and reduces ghosting by adapting the blend weights to the accumulated texture deformation. Kwatra et al [2005] cast synthesis as a global optimization over an overlapping set of blended neighborhoods. They achieve advection by warping the result of the previous frame with the flow field, and using the warped image as a soft constraint when optimizing the current frame.

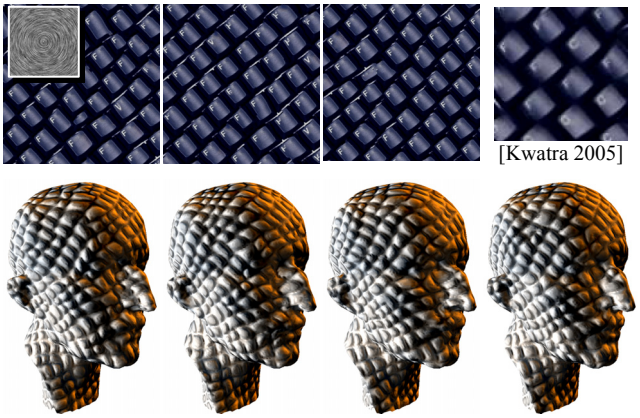


Figure 13: Results of texture advection in 2D and on surfaces. Paradoxically, static frames from an ideal result may reveal little about the underlying flow field. So, seeing the video is crucial.

Our approach combines ideas from both these prior techniques. Given a velocity field $V(p)$ in domain D, by default we simply advect the synthesized *coordinates* of the previous frame $t-1$ to obtain the result at the current frame t. We replace the synthesized coordinates in-place as $S^t[p] := S^{t-1}[p] + J(p)\,V(p)$.

Although transforming the synthesized coordinates creates a temporally smooth result, the texture gradually distorts in areas of flow divergence. Therefore, we must "regenerate" the texture using synthesis correction. However, achieving coherent synthesis requires upsampling parent pixels within the coarse-to-fine pyramid, which can increase temporal discontinuities. As a tradeoff between temporal coherence and exemplar fidelity, we upsample from the coarser level only in areas where the distortion of the synthesized texture exceeds a threshold. We measure distortion as the Frobenius norm $\xi = \|J_S - J\|_2$ between the

observed Jacobian $J_S = (\mathrm{ddx}(S)\ \mathrm{ddy}(S))$ of the synthesized texture and the desired anisometric Jacobian J (defined in Sections 5-6). Thus, the upsampling pass becomes

$$S_l^t[p] := \begin{cases} S_l^{t-1}[p] + J(p)V(p) & , \xi(p) < c \\ S_{l-1}^t[p - \Delta] + J(p)\Delta & , otherwise \end{cases}.$$

As an optimization, we find that obtaining good advection results only requires processing the 3-4 finest synthesis levels.

Compared to [Kwatra et al 2005], our advecting textures can conform to an anisometric field to allow flow of undistorted features over an arbitrary surface. Semantic features such as the keys and pustules in Figure 13 advect without blurring. And, synthesis is 3 orders of magnitude faster.

8. Nonlinear dimensionality reduction

Because exemplar transformation is a preprocess, we can replace linear PCA by nonlinear dimensionality reduction without affecting the performance of runtime synthesis. We have explored two such techniques: isomaps [Tenenbaum et al 2000] and locally linear embedding (LLE) [Roweis and Saul 2000].

Both isomaps and LLE aim to parameterize the data over a nonlinear manifold. They approximate the local structure of this manifold by building a weighted graph on the points using either a global distance threshold or *k*-nearest neighborhoods. We have found this graph construction to be challenging in our problem setting. Distance thresholds become unstable in high-dimensional spaces due to low variance in distances. And, *k*-neighborhoods behave poorly due to the presence of dense degenerate clusters. These clusters are in fact textons – groups of points with similar neighborhoods [Malik et al 1999]. Therefore, we perform fine clustering as a preprocess to collapse degenerate clusters, prior to constructing a k=70 neighborhood graph on this regularized data.

We experiment with 4D transformed exemplars to emphasize differences (Figure 14). We find that isomaps lead to better texture synthesis results than LLE. One explanation is that isomaps are less likely to map dissimilar neighborhoods to nearby points in the transformed exemplar space, because they preserve geodesic distances between all pairs of points, whereas LLE preserves the geometry of local neighborhoods.

So far, isomap results are comparable to those of PCA, perhaps with a slight improvement. We think there is unique opportunity to further adapt and extend sophisticated nonlinear dimensionality reduction techniques to improve neighborhood comparisons while still enabling real-time synthesis.

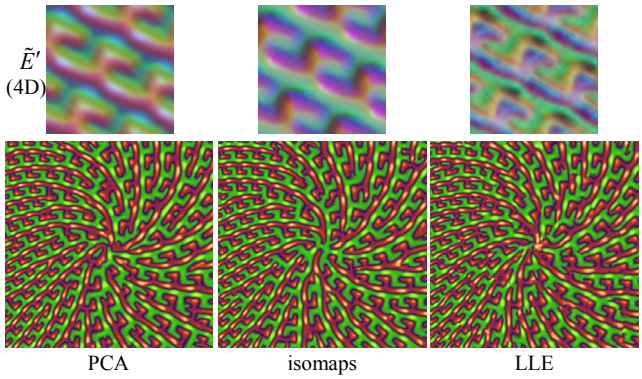


Figure 14: Comparison of appearance-space dimensionality reduction using PCA, isomaps, and LLE, and resulting synthesis.

9. Discussion and additional results

Recall that we perform PCA projection twice: for appearance-space dimensionality reduction $E' \to \tilde{E}'$ and for runtime neighborhoods $N(\tilde{E}') \to \tilde{N}(\tilde{E}')$. We can quantify the effectiveness of these projections by computing their fractional residual variance. Figure 15 plots appearance-space residual variance as a function of the dimension of the transformed exemplar $\tilde{E}'$. Each curve corresponds to a different level of coarse-to-fine synthesis (6 is finest) on the Figure 3 exemplar. For this dataset, the most challenging level is 3, where the 8D transformed exemplar loses 21% of the total variance. In some sense, this resolution level has the most complex spatial structure.

Figure 16 compares such curves for a simple color texture, a texture with a signed-distance feature channel, and a radiance-transfer texture. As expected, these texture types have appearance-space distributions that are progressively more complex. Table 1 summarizes this for the textures we have tested.

The results suggest that appearance-space dimensionality reduction can lose significant information and still permit effective texture synthesis. It is interesting to put this in the context of traditional synthesis schemes, in which appearance at an exemplar location is estimated by just point-sampling color. Intuitively, these schemes provide a constant-color approximation in our appearance space. We find empirically that this constant-color approximation has a mean squared error that is about 5-12 times larger than our 8D PCA residual variance. In effect, the larger runtime neighborhood comparisons used in earlier synthesis schemes helped compensate for this missing information.

Pixel-based schemes often use a parameter κ to artificially favor coherent patches [Hertzmann et al 2001]. We find that this bias becomes much less important in appearance-space synthesis. The bias is only beneficial in extreme cases such as undersampled surface regions and areas of rapidly changing Jacobian.

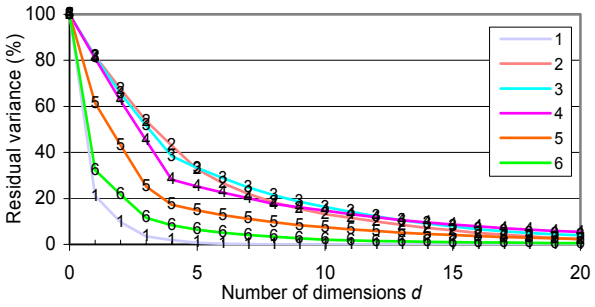


Figure 15: Appearance-space variance unaccounted by the largest d=1…20 principal components, for synthesis levels 1-6.

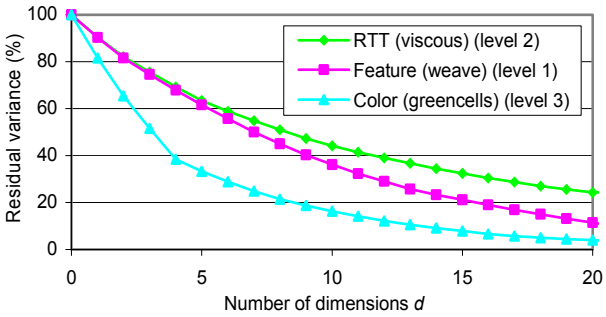


Figure 16: Comparison of appearance-space residual variance for a color texture, a texture with feature distance, and an RTT.

Data type	E′ dim.	PCA residual variance (max over levels)			
		8D $\tilde{E}'$		8D $\tilde{N}(\tilde{E}')$	
		mean	sdv	mean	sdv
RGB color	75	26%	9%	25%	11%
RGB + feature	100	30%	10%	27%	12%
RTT	900	36%	16%	19%	12%

Table 1: Fraction of variance lost in the two PCA projections, expressed as mean and standard deviation over all datasets.

Synthesis mode	Synthesis rate (frames/sec)	
	Standard size E:64^2, S:256^2	Large size E:128^2, S:512^2
2D isometric	48.3	8.4
2D anisometric	40.4	8.1
Surface atlas	54.7	13.1
Advection over surface	88.6	19.7

Table 2: Runtime performance in frames per second, including synthesis and rendering with magnification.

All results are obtained with Microsoft DirectX 9 on an NVIDIA GeForce 7800 with 256MB memory. Texture atlases are created using DirectX UVAtlas. For 2D isometric synthesis, the number of pixel shader instructions in the upsampling and correction passes is 45 and 383 respectively. When including all functionalities (anisometry, atlas indirection, advection), these increase to 52 and 516 instructions respectively. For each pyramid synthesis level, we perform 2 correction passes, each with 4 subpasses.

Table 2 summarizes runtime synthesis performance for different exemplar and output sizes. As demonstrated on the video, we can manipulate all synthesis parameters interactively since the texture is regenerated every frame.

10. Summary and future work

We transform an exemplar into an appearance space prior to texture synthesis. This appearance space is low-dimensional (8D) and Euclidean, so we avoid the large (e.g. 400^2) inner-product matrices of texton schemes, as well as any noise due to discrete texton quantization. By including spatial neighborhood, semantic features, and radiance-transfer into the appearance vectors, we achieve results similar to earlier specialized schemes, but with a simpler, unifying framework that is several orders of magnitude faster and extends easily to anisometric synthesis and advection.

Pixel-based approaches are often perceived as inherently limited due to narrow neighborhoods and lack of global optimization. In this regard, results such as Figure 8 have unexpected quality. The robustness of appearance-space synthesis is most evident in our advection results, where the added constraint of temporal coherence makes synthesis particularly challenging.

There are a number of avenues for future work:

- Consider other appearance-space attributes, such as foreground-background segmentation in multi-layer textures.
- Synthesize view-dependent RTT or BTF. We believe that this should still be possible with an 8D transformed exemplar because the texture mesostructure is already captured accurately.
- Further explore nonlinear dimensionality reduction.
- Consider spatiotemporal neighborhoods for video textures.

Acknowledgments

We thank Ben Luna, Peter-Pike Sloan, and John Snyder for providing the RTT datasets and libraries.

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. *Symposium on Interactive 3D Graphics*, 217-226.

DE BONET, J. 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. *ACM SIGGRAPH*, 361-368.

EFROS, A., AND LEUNG, T. 1999. Texture synthesis by non-parametric sampling. *ICCV*, 1033-1038.

GARBER, D. 1981. Computational models for texture analysis and texture synthesis. PhD Dissertation, University of Southern California.

HEEGER, D., AND BERGEN, J. 1995. Pyramid-based texture analysis/synthesis. *ACM SIGGRAPH*, 229-238.

HERTZMANN, A., JACOBS, C., OLIVER, N., CURLESS, B., AND SALESIN, D. 2001. Image analogies. *ACM SIGGRAPH*, 327-340.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. *ACM SIGGRAPH*, 517-526.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *SIGGRAPH*, 795-802.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM SIGGRAPH*, 777-786.

LEUNG, T., AND MALIK, J. 2001. Representing and recognizing the visual appearance of materials using 3D textons. *IJCV* 43(1), 29-44.

LIANG, L., LIU, C., XU, Y., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM TOG* 20(3), 127-150.

MAGDA, S., AND KRIEGMAN, D. 2003. Fast texture synthesis on arbitrary meshes. *Eurographics Symposium on Rendering*, 82-89.

MALIK, J., BELONGIE, S., SHI, J., AND LEUNG, T. 1999. Textons, contours and regions: Cue integration in image segmentation. *ICCV*, 918-925.

NEYRET, F., AND CANI, M.-P. 1999. Pattern-based texturing revisited. *ACM SIGGRAPH*, 235-242.

NEYRET, F. 2003. Advected textures. *Symposium on computer animation*, 147-153.

POPAT, K., AND PICARD, R. 1993. Novel cluster-based probability model for texture synthesis, classification, and compression. *Visual Communications and Image Processing*, 756-768.

PORTILLA, J., AND SIMONCELLI, E. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *IJCV* (40)1.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. *ACM SIGGRAPH*, 465-470.

ROWEIS, S. 1997. EM algorithms for PCA and SPCA. *NIPS*, 626-632.

ROWEIS, S., AND SAUL, L. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323-2326.

SLOAN, P.-P., LIU, X., SHUM, H.-Y., AND SNYDER, J. 2003. Bi-scale radiance transfer. *ACM SIGGRAPH*, 370-375.

TAPONECCO, F., AND ALEXA, M. 2004. Steerable texture synthesis. *Eurographics Conference*.

TENENBAUM, J., DE SILVA, V., AND LANGFORD, J. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319-2323.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM SIGGRAPH*, 665-672.

TURK, G. 2001. Texture synthesis on surfaces. *SIGGRAPH*, 347-354.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. *ACM SIGGRAPH*, 479-488.

WEI, L.-Y., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. *ACM SIGGRAPH*, 355-360.

WEI, L.-Y., AND LEVOY, M. 2003. Order-independent texture synthesis. http://graphics.stanford.edu/papers/texture-synthesis-sig03/.

WU, Q., AND YU, Y. 2004. Feature matching and deformation for texture synthesis. *ACM SIGGRAPH*, 362-365.

YING, L., HERTZMANN, A., BIERMANN, H., AND ZORIN, D. 2001. Texture and shape synthesis on surfaces. *Symposium on Rendering*, 301-312.

ZHANG, J., ZHOU, K., VELHO, L., GUO, B., AND SHUM, H.-Y. 2003. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM SIGGRAPH*, 295-302.