Solid Texture Synthesis from 2D Exemplars

Johannes Kopf, University of Konstanz
Chi-Wing Fu, Hong Kong University of Science and Technology
Daniel Cohen-Or, Tel Aviv University
Oliver Deussen, University of Konstanz
Dani Lischinski, The Hebrew University
Tien-Tsin Wong, The Chinese University of Hong Kong

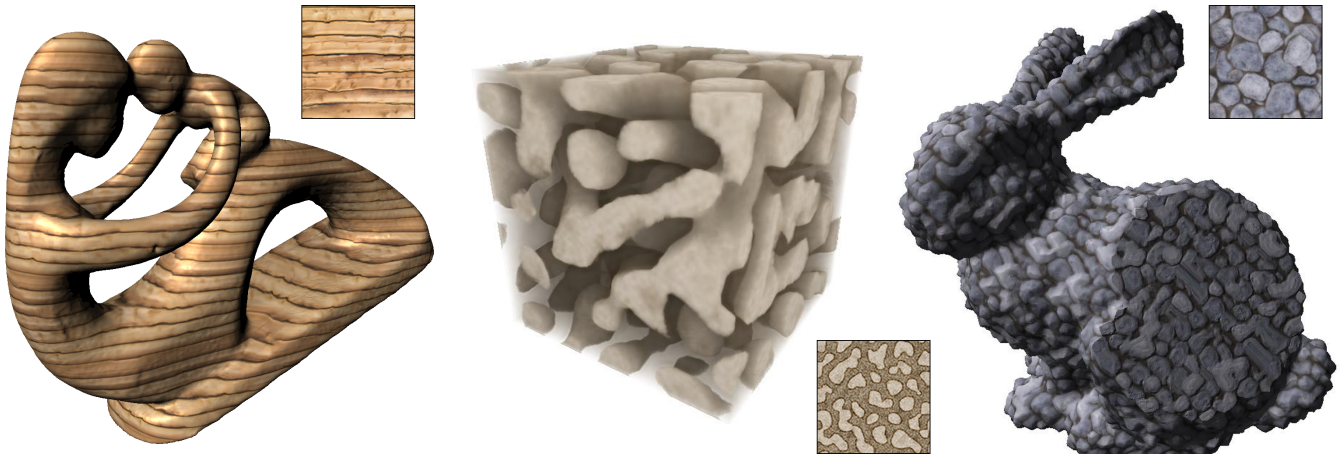

Figure 1: Examples of solid textures synthesized with our approach. Left: the statue appears to be carved out of a block of wood. Middle: volume rendering this solid texture with the brown texels rendered as transparent reveals intricate internal structure. Right: cutting off a part of the bunny reveals a consistent stone texture in the interior (we *synthesized* a displacement channel along with the RGB channels). The input 2D exemplars are shown next to the solid textured models.

Abstract

We present a novel method for synthesizing solid textures from 2D texture exemplars. First, we extend 2D texture optimization techniques to synthesize 3D texture solids. Next, the non-parametric texture optimization approach is integrated with histogram matching, which forces the global statistics of the synthesized solid to match those of the exemplar. This improves the convergence of the synthesis process and enables using smaller neighborhoods. In addition to producing compelling texture mapped surfaces, our method also effectively models the material in the interior of solid objects. We also demonstrate that our method is well-suited for synthesizing textures with a large number of channels per texel.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: texture synthesis, solid texture

1 Introduction

Texture mapping is one of the most essential techniques for realistic image synthesis, since it enables augmenting geometric models with rich and realistic visual detail. Texture synthesis techniques are thus of much interest. In this work we present a new method for synthesizing a *3D solid texture* from a 2D exemplar.

Solid textures [Peachey 1985; Perlin 1985] have several notable advantages over 2D textures. First, many natural materials, such as wood and stone, may be more realistically modeled using solid textures (see Figure 1). Second, solid textures obviate the need for finding a parameterization for the surface of the object to be textured, which is a challenging problem in itself. In fact, for objects of general topology it is not possible to find a parameterization that avoids seams and/or distortion. Although these problems may be alleviated by synthesizing directly on the surface of an object (e.g., [Turk 1991; Turk 2001; Wei and Levoy 2001; Ying et al. 2001]), they cannot be avoided altogether.

Furthermore, solid textures provide texture information not only on surfaces, but also throughout the entire volume occupied by a solid object. This is a highly convenient property, as it makes it possible to perform high-fidelity sub-surface scattering simulations, as well as break objects to pieces and cut through them, as demonstrated in Figure 1.

So far, solid textures have (almost exclusively) been generated procedurally. Procedural textures are attractive, because they compactly represent solid textures with unbounded spatial extent and resolution. However, they can also be difficult to control. In particular, there's no general automatic way of developing a procedure that convincingly reproduces some specific natural pattern. This challenging task is left to the user, and typically requires considerable expertise and trial-and-error. This makes example-based synthesis an appealing alternative.

However, 3D texture exemplars are difficult to obtain, while synthesizing a 3D solid texture from a 2D exemplar is in many respects an extremely challenging task. For example, some of the most effective 2D synthesis methods directly copy patches of texture from the exemplar to the result. Since the synthesized texture looks just

ACM Reference Format
Kopf, J., Fu, C., Cohen-Or, D., Deussen, O., Lischinski, D., Wong, T. 2007. Solid Texture Synthesis from 2D Exemplars. *ACM Trans. Graph. 26*, 3, Article 2 (July 2007), 9 pages. DOI = 10.1145/1239451.1239453 http://doi.acm.org/10.1145/1239451.1239453.

like the exemplar inside each patch, the synthesis process need only to worry about hiding the seams between the patches. In contrast, when the result has an additional dimension, *every slice* through *each texel* in the volume has to look like the exemplar, and they all need to agree with each other.

Time complexity is also a major concern for solid texture synthesis, as the texel count grows cubically with the spatial resolution. Thus, algorithms that are, e.g., of quadratic complexity are out of the question for solid texture synthesis.

In this work we use a non-parametric global texture optimization approach [Wexler et al. 2004; Kwatra et al. 2005] to synthesize 3D texture solids. This is done by simultaneously minimizing an MRF-based distance between several orthogonal 2D neighborhoods around each voxel of the 3D texture and their corresponding neighborhoods in the 2D exemplar. We also introduce a novel reweighting scheme, inspired by Heeger and Bergen's seminal work [1995], which ensures that the result preserves the global statistics of the exemplar. Thus, we achieve both histogram matching and neighborhood matching, leveraging the strengths of both parametric and non-parametric texture synthesis methods.

The result is a new method for synthesis of solid textures from 2D exemplars, which is generally superior to previous methods in terms of the quality of the synthesized results. More importantly, it is applicable to a wider variety of textures, without sacrificing computational efficiency. We show that the method is effective for texturing both the surface and the interior of solid objects. We also demonstrate that our method is well-suited for synthesizing multi-channel textures, with or without correlation between the channels. For example, we can synthesize a variety of surface attributes (in addition to surface color).

2 Related Work

During the past decade many example-based texture synthesis methods have been proposed. Over the years we have witnessed a shift from parametric methods [Heeger and Bergen 1995], to non-parametric methods [De Bonet 1997], including pixel-based methods [Efros and Leung 1999; Wei and Levoy 2000], patch-based methods [Efros and Freeman 2001; Kwatra et al. 2003], and most recently to optimization-based methods [Kwatra et al. 2005], and appearance-space texture synthesis [Lefebvre and Hoppe 2006]. Parametric methods attempt to construct a parametric model of the texture based on the input sample, which has proven to be a challenging task, and are mostly successful with homogeneous and stochastic textures. Non-parametric methods have demonstrated the ability to handle a much wider variety of textures, by growing the texture one pixel/patch at a time. Optimization-based methods evolve the texture as a whole, further improving the quality of the results and making the synthesis more controllable. In this work we integrate texture optimization with preservation of some of the input texture's global statistical properties; this both speeds up convergence and helps to avoid undesirable local minima.

Example-based 2D texture synthesis methods have also been extended to synthesize 3D textures from 3D input samples. This has been explored mostly with the third dimension being time [Szummer and Picard 1996; Schödl et al. 2000; Wei and Levoy 2000; Bar-Joseph et al. 2001; Soatto et al. 2001; Kwatra et al. 2003]. Solid textures could probably also be generated in this manner. However, digitized 3D texture samples of real materials are difficult to obtain, especially for multi-channel textures, such as BTFs, while 2D texture samples are abundant.

Applying a 2D texture to an object of arbitrary topology requires finding a parameterization. Hence, several researchers explored the idea of synthesizing the texture directly on the surface of the target object. Wei and Levoy extend their texture synthesis method [2000] by generalizing their definition of neighborhood search in order to work on general surfaces [2001]. Turk [2001] uses a hierarchy of meshes in combination with a user-specified vector field. Tong *et al.* [2002] and later Liu *et al.* [2004] also synthesize BTFs directly on arbitrary surfaces.

Shell texture functions [Chen et al. 2004] apply a relatively thin shell on the object surface to speed up rendering of complex translucent materials. The shell texture is synthesized using the approach of Tong *et al.* [2002]. Complex inner structures cannot be represented here, since the inner core of the material is modeled as homogeneous.

While the above methods have been able to produce convincing results, they require re-synthesizing the texture from scratch for each target object. In contrast, once a solid texture is available, it may be applied to any 3D object, and the only remaining costs are those of rendering the textured object.

The earliest pioneering attempts to synthesize solid textures from 2D texture samples used a parametric approach. For example, Ghazanfarpour and Dischler [1995; 1996] attempt to match the spectral characteristics of a 2D texture sample, while Heeger and Bergen [1995] propose a multi-scale histogram matching approach. Dischler *et al.* [1998] combine spectrum and histogram matching and use orthogonal 2D views in order to synthesize anisotropic solid textures. These methods are designed to handle textures whose appearance is well captured by global parameters, such as the frequency spectrum and/or histogram, and thus cannot generally handle the large class of natural textures exhibiting macro-structures.

Wei [2002; 2003] made the first attempt to extend his non-parametric 2D synthesis method to synthesize solid textures. His results demonstrate the difficulty of such an extension: they exhibit considerable blurring and are unable to preserve even fairly small structures. In this work we are able to demonstrate better results by extending a more recent global texture optimization approach.

Qin and Yang [Qin and Yang 2007] propose a method for synthesis of solid textures using Basic Gray Level Aura Matrices (BGLAMs), which are based on concepts originally introduced by Elfadel and Picard [1994]. The BGLAMs of a texture characterize the cooccurrence probability distributions of gray levels at all possible displacement configurations. While some of the presented results are impressive, the approach has a significant drawback: the basic method works only on grayscale images. To process color textures, the color channels must be decorrelated as proposed by [Heeger and Bergen 1995]. However, in most textures the color channels are strongly correlated, and independently synthesizing the decorrelated channels leads to visual artifacts, as demonstrated in Figure 2. It should be noted that constructing aura matrices that capture the cooccurrence probabilities of multi-channel texels is not a feasible solution, since the size of this representation grows quadrati-

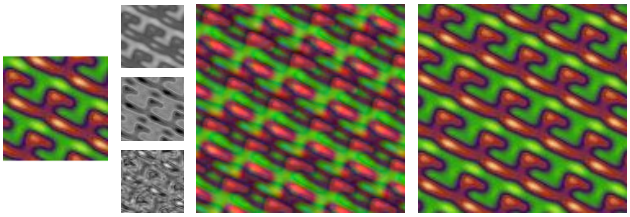

Figure 2: Independent color channel synthesis. (a) Input texture, (b) decorrelated color channels, computed as described by Qin and Yang [2007]. Note that the channels are still correlated and independent channel synthesis results in visual artifacts (c). (d) Our result, where the channels are synthesized together.

cally in the number of distinct colors in the texture. Thus, the aura matrices would be huge on the one hand, but very sparse and not containing enough statistics, on the other hand. In contrast, our new method is able to handle correlated color textures, as well as other multi-channeled textures, in which the channels are also often highly correlated.

Another approach that was explored is to estimate the parameters of a procedural shader so as to match a given texture sample [Lefebvre and Poulin 2000; Qin and Yang 2002]. This approach may be used to generate a solid texture based on a 2D example, but assumes that the shader is already available and only its parameters are unknown.

Finally, Jagnow *et al.* [2004] present an approach for solid texture synthesis, based on stereology techniques. This is a statistical approach applicable to materials composed of particles embedded in a binding medium. However, this approach requires having models for the different particle shapes that may be present in the solid. We show that our method is capable of delivering results of similar visual fidelity, but in a more automatic fashion and on a much wider variety of textures.

3 Overview

Our method for synthesizing solid textures from 2D texture exemplars integrates ideas and techniques from non-parametric texture synthesis together with a global histogram matching approach.

In the next section we describe how to extend global texture optimization [Kwatra et al. 2005; Wexler et al. 2007] to the task of solid texture synthesis. While the basic optimization framework is not new, the challenging nature of the task demanded that we choose and tune the parameters in a judicious manner.

The goal of the optimization process is to minimize a global texture energy function that measures the extent to which the synthesized solid texture deviates from the exemplar over a variety of local 2D neighborhoods. However, there is a danger that such a process could get stuck in a local minimum; for example, repeating the same exemplar neighborhoods over and over again, and failing to make use of the full richness of the exemplar. In order to address this issue, we integrate the optimization process with a histogram matching scheme, described in Section 5. Histogram matching ensures that the synthesized solid is similar to the exemplar not only over local neighborhoods, but also in its global statistics. We found that this often results in solids that are more similar in appearance to the exemplars. Furthermore, histogram matching significantly improves performance by making the convergence of the optimization process much faster, as well as allowing the use of relatively small fixed size neighborhoods (8×8). It should be noted that these improvements in quality and performance apply not only to solid synthesis, but also to 2D texture synthesis.

Although for the sake of simplicity we sometimes refer in this paper to the texels and voxels as having colors, none of the algorithms or the equations below are limited to three channel RGB textures. Rather, each texel can hold a high-dimensional vector.

4 Solid Optimization

The solid optimization process begins with a volume where the value of each voxel is randomly chosen from the exemplar. The goal is to iteratively increase the similarity between the solid texture and the exemplar by minimizing an energy function that measures the differences between the two. Specifically, for isotropic solid textures we would like every local neighborhood on any 2D slice through the 3D solid to be similar to some neighborhood in the exemplar. In order to reduce computation time and to avoid resampling issues, we only measure the differences on the three slices orthogonal to the main axes of the volume, as in previous approaches

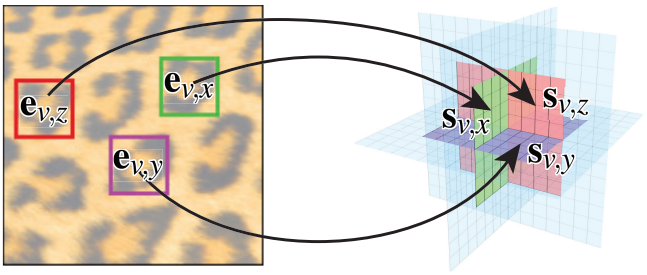

Figure 3: The neighborhoods used in our equations.

[Wei 2002; Qin and Yang 2007]. As we shall see, this approximation works well in practice, and the resulting solid textures are similar to the exemplar on arbitrary slices through the volume.

Denoting by $\mathbf{e}$ the input exemplar, and by $\mathbf{s}$ the synthesized solid, the global texture energy that we seek to minimize is defined as

$$E(\mathbf{s},\{\mathbf{e}\}) = \sum_{v} \sum_{i\in\{x,y,z\}} \left\|\mathbf{s}_{v,i} - \mathbf{e}_{v,i}\right\|^{r}. \tag{1}$$

Here $\mathbf{s}_v$ refers to a single voxel, and $\mathbf{s}_{v,x}$, $\mathbf{s}_{v,y}$, and $\mathbf{s}_{v,z}$ are the vectorized neighborhoods of v in the slices orthogonal to the x, y, and z axis, respectively, as shown in Figure 3. The exemplar neighborhood closest to $\mathbf{s}_{v,i}$ (in L_2 norm) is denoted by $\mathbf{e}_{v,i}$. The exponent $r = 0.8$ causes the optimization to be more robust against outliers [Kwatra et al. 2005].

The texture energy is minimized in an iterative fashion, alternating between two phases. In the *optimization phase* we update the value of each voxel $\mathbf{s}_v$, based on the best matching neighborhoods of the neighboring voxels. In the *search phase* we search for the best matching exemplar neighborhoods $\mathbf{e}_{v,i}$ for each voxel $\mathbf{s}_{v,i}$. The two phases are described in more detail in the remainder of this section. The process is carried out in a multiresolution fashion: we start with a coarse version of the volume, using trilinear interpolation to switch to a finer level once the coarser level has converged.

4.1 Optimization phase

We use iteratively re-weighted least squares (IRLS), similarly to Kwatra *et al.* [2005], to minimize the energy. To this end, we rewrite the terms of the energy functional (1) as follows:

$$\left\|\mathbf{s}_{v,i} - \mathbf{e}_{v,i}\right\|^{r} = \underbrace{\left\|\mathbf{s}_{v,i} - \mathbf{e}_{v,i}\right\|^{r-2}}_{\omega_{v,i}} \left\|\mathbf{s}_{v,i} - \mathbf{e}_{v,i}\right\|^{2} = \omega_{v,i} \left\|\mathbf{s}_{v,i} - \mathbf{e}_{v,i}\right\|^{2} \tag{2}$$

and minimize the following quadratic functional:

$$E(\mathbf{s},\{\mathbf{e}\}) = \sum_{v} \sum_{i\in\{x,y,z\}} \sum_{u\in N_i(v)} \omega_{v,i,u} \left(\mathbf{s}_{v,i,u} - \mathbf{e}_{v,i,u}\right)^{2} \tag{3}$$

Here, $N_i(v)$ denotes the neighborhood of the voxel v in the slice perpendicular to the i-th axis, and (for now) $\omega_{v,i,u} = \omega_{v,i}$. Note that each voxel participates in a number of terms: one for each neighborhood it belongs to. Assuming that the weights $\omega_{v,i,u}$ are constant during the optimization phase, and setting the derivative of (3) with respect to $\mathbf{s}_v$ to zero yields the following solution:

$$\mathbf{s}_v = \frac{\sum_{i\in\{x,y,z\}} \sum_{u\in N_i(v)} \omega_{u,i,v}\, \mathbf{e}_{u,i,v}}{\sum_{i\in\{x,y,z\}} \sum_{u\in N_i(v)} \omega_{u,i,v}}. \tag{4}$$

Here, $\mathbf{e}_{u,i,v}$ denotes the exemplar texel in the neighborhood $\mathbf{e}_{u,i}$ that corresponds to v. Thus, the optimal value of each voxel is simply a

weighted average of a collection of texels from different exemplar neighborhoods.

In practice, computing the weighted average using (4) may sometimes produce blurry results, if the variance of the exemplar texels $\mathbf{e}_{u,i,v}$ is too large. In such cases, we employ a clustering approach, proposed by Wexler *et al.* [2007]. Treating the texels $\mathbf{e}_{u,i,v}$ in equation (4) as points in a high-dimensional space, we cluster them using the Mean-Shift algorithm. We then average only those exemplar texels that belong to the dominant cluster.

The bandwidth (window size) of the Mean-Shift algorithm is gradually reduced as the optimization of each resolution level progresses. Large bandwidth results in larger clusters, containing many or all of the colors, and the result is similar to plain averaging. As the bandwidth is reduced, the dominant cluster becomes smaller, in turn reducing the blurring.

4.2 Search phase

In this phase we optimize (1) with respect to $\mathbf{e}_{v,i}$ by finding the best matching exemplar window for every neighborhood $\mathbf{s}_{v,i}$. This is a standard nearest neighbor search in a high-dimensional space, and it dominates the running time of our optimization.

We speed this step up in a number of ways. First, we apply a PCA projection to the neighborhood vectors in the exemplar [Hertzmann et al. 2001; Liang et al. 2001; Lefebvre and Hoppe 2006]. We keep only the number of coefficients sufficient to preserve 95% of the variance. For RGB textures with 8×8 neighborhoods the dimensionality is usually reduced from 192 to about 10–30 dimensions, depending on the size and richness of the exemplar. Thus, performance is improved drastically. For multi-channel textures, such as BTFs, initial experiments show the improvements to be even more dramatic, with the dimensionality dropping from tens of thousands to a couple of hundreds.

The dimensionality reduction paves the way for using approximate nearest neighbor techniques, which are far more efficient for the resulting relatively low dimensional spaces. We use the ANN approximate nearest neighbor library [Mount and Arya 2006]. ANN takes as a parameter an ε value, and returns an approximate nearest neighbor that lies no farther than $(1+\varepsilon)$ times the distance to the true nearest neighbor. We found in our experiments $\varepsilon = 2$ to be a good compromise between speed and accuracy.

Finally, we do not search for the best neighborhoods at every voxel, but rather on a sparser grid. More specifically, for slices orthogonal to the x axis, we perform the search only for a sparser set of voxels $g_x = \{(i, m\cdot j, m\cdot k), \forall i,j,k\}$, and similarly for the y and z axes. For all the results in this paper we used $m = 2$.

5 Histogram Matching

For many textures the optimization process described in the previous section could converge to a wrong local minimum, because the energy function measures only the similarity of local neighborhoods, without accounting for any global statistics. This is demonstrated by the first and third rows of images in Figure 4. The converged results achieve low texture energy because the local neighborhoods fit well together, yet the result does not look quite similar to the exemplar; only a small number of neighborhoods from the exemplar participate in the result and it does not reflect the full richness of the exemplar.

We address this problem by introducing a re-weighting scheme designed to make certain global statistics of the resulting texture to remain close to those of the exemplar. More specifically, we carefully adjust the weights in eq. (4) so as to effectively ensure that certain histograms of the synthesized texture match the exemplar.

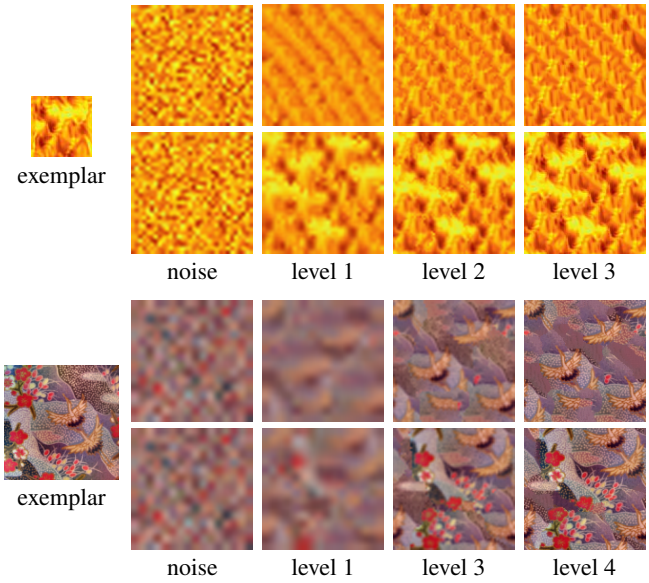

Figure 4: The convergence of the texture optimization process without (rows 1 and 3) and with (rows 2 and 4) histogram matching. For each method we show the initial noise texture followed by the result at the end of each resolution level.

During the optimization phase we construct and keep track of one 16-bin histogram for each of the texture's channel, e.g., for each of the R, G, and B channels in the case of a color texture. In principle it is also possible to use other histograms, such as histograms of gradients or of steerable coefficients, as done by Heeger & Bergen [1995]. Whenever a value of a voxel is to be updated according to eq. (4), we examine all of the exemplar texels that participate in this weighted average and *reduce the weight* of each texel that might contribute to an *increase* in the difference between a histogram of the result and the corresponding histogram of the exemplar.

More formally, let $H_{\mathbf{s},j}$ and $H_{\mathbf{e},j}$ denote the j-th histogram of the synthesized result and the exemplar, respectively, and let $H(b)$ denote the value of bin b in a histogram H. Next, for a color c, let $b_j(c)$ specify the bin containing c in the histograms $H_{\mathbf{s},j}$ and $H_{\mathbf{e},j}$. We modify the weights for equation (4) in the following way:

$$\omega'_{u,i,v} = \frac{\omega_{u,i,v}}{1+\sum_{j=1}^{k} \max\left[0, H_{\mathbf{s},j}(b_j(\mathbf{e}_{u,i,v})) - H_{\mathbf{e},j}(b_j(\mathbf{e}_{u,i,v}))\right]} \quad (5)$$

Intuitively speaking, the above equation reads as follows. If an exemplar texel $\mathbf{e}_{u,i,v}$ has a large weight in the average, it "pulls" the synthesized texel $\mathbf{s}_v$ to the bin $b = b_j(\mathbf{e}_{u,i,v})$ in the result histogram. If the result histogram has a smaller count than the exemplar histogram in this bin ($H_{\mathbf{s},j}(b) < H_{\mathbf{e},j}(b)$), this is desirable, because increasing the count $H_{\mathbf{s},j}(b)$ would make it closer to $H_{\mathbf{e},j}(b)$. However, in the contrary case that $H_{\mathbf{e},j}(b) < H_{\mathbf{s},j}(b)$, increasing the count would increase the difference between the two histograms. In such a case, we interfere by reducing the weight assigned to $\mathbf{e}_{u,i,v}$.

Histogram matching causes the *global* statistics of the texture to match the exemplar, while the neighborhood matching terms of the optimization enforce *local* similarities. The integrated approach automatically adapts itself to the current situation: If the synthesis histograms are far off the exemplar ones we effectively prevent "bad texels" from contributing to the synthesized value, whereas if the histograms are close, the weights are largely unaffected and the synthesis turns to neighborhood-matching only.

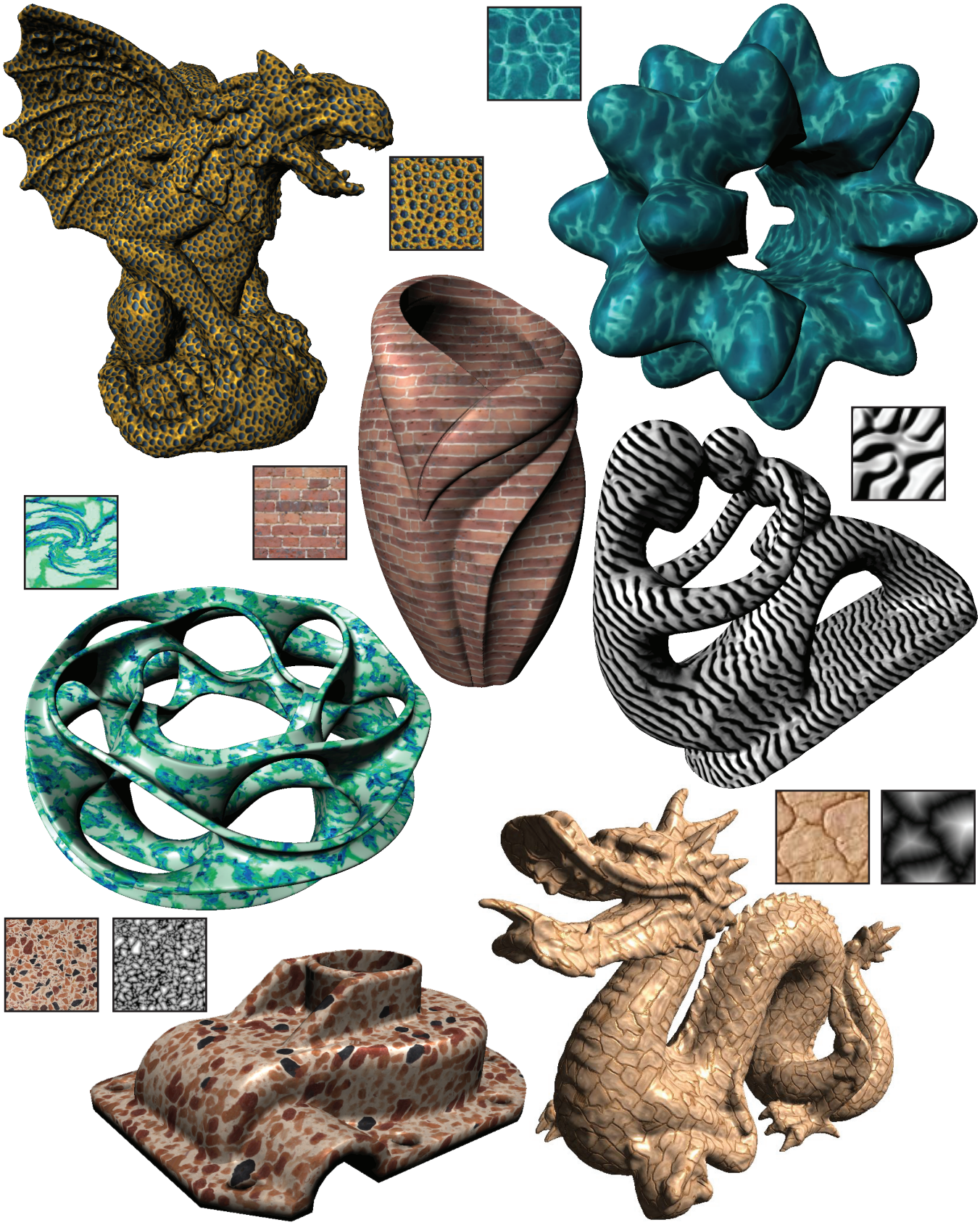

Figure 5: Various solid textures generated with our method, applied to objects with complex geometry and topology. The 2D exemplar (and the feature map, when one was used) is shown next to each texture mapped object. Note the anisotropy in some of the textures.

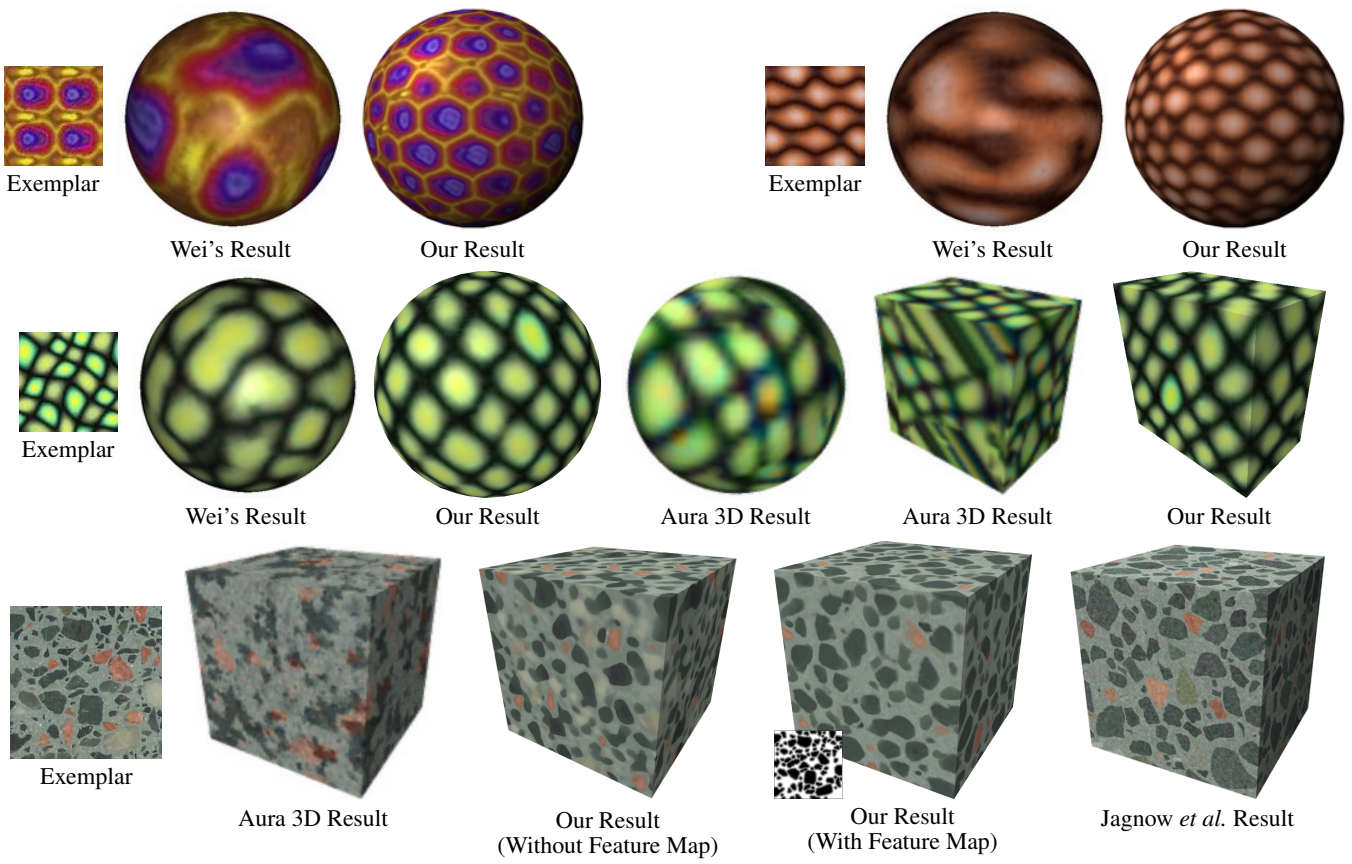

Figure 6: Comparison with previous methods.

Our experiments have shown that the histograms must be kept up to date as the synthesis progresses. For example, if the histograms are only updated once before each iteration, all neighborhoods get reweighted in the same manner, which causes overshooting and divergence of the algorithm. Therefore, to keep track of the evolving result histograms we update them simultaneously with the voxel updates. To avoid any directional bias, we visit the voxels in a random order.

We found histogram matching to dramatically improve the convergence of the algorithm (see the supplementary video). Moreover, it allows us to successfully synthesize textures using small, fixed-sized neighborhoods. Specifically, we use 8×8 windows, whereas previous methods reported using much larger neighborhoods (32×32 and larger) to recreate the features of the exemplar. The smaller neighborhoods cause our algorithm to be much faster, which is crucial since for solid textures we deal with a very large number of voxels and long runtimes in general.

6 Results

We implemented our approach in C++ and used it to generate a wide variety of solid textures from 2D exemplars. In all of our experiments we use a three-level synthesis pyramid, with a fixed-size 8×8 neighborhood at each level. The synthesis times depend on the size and richness of the exemplar textures. Synthesizing a 128^3 solid texture with 3 color channels per texel takes anywhere between 10 and 90 minutes on a 2.4GHz CPU.

Figure 5 shows some representative results of synthesized solid RGB textures, effortlessly mapped on a variety of 3D objects with non-trivial geometry and topology. Additional results are available in the supplemental materials and on the accompanying video. It should be noted that although some of the input textures do not necessarily correspond to a 2D slice through a solid material, our solid synthesis approach is still able to generate a very plausible solid texture, which looks quite convincing when applied to these (and many other) 3D objects.

Even though the formulation of the global texture energy (1) assumes isotropic textures, it may be seen that some of our results are actually quite anisotropic (for example, the dunes and the bricks solid textures in Figure 5). Note that in these cases, no isotropic extension of the exemplar to 3D exists! Nevertheless, our method automatically produces a highly plausible solid texture. Although we attempt to match the exemplar on all three orthogonal slices through each voxel, it appears that two out of the three directions become dominant and succeed in matching the exemplar. Slices corresponding to the third direction are less similar to the exemplar, but are still coherent and plausible.

It has been shown that some textures can be synthesized better with the aid of feature maps, which provide non-local feature information [Wu and Yu 2004; Lefebvre and Hoppe 2006]. We found feature maps to be particularly helpful for textures with strong large structures, such as stone patterns. In such cases we provide our method with a feature map as an extra channel. Our feature maps encode the signed feature distance, similarly to [Lefebvre and Hoppe 2006]. When showing results produced using a feature map we show the map next to the exemplar (except in Figure 1).

Figure 6 shows a comparison with several previous approaches (using images extracted from the corresponding papers and/or websites). It may be observed that our results do not suffer from blurring and preserve structures better than those by Wei [2002]. The

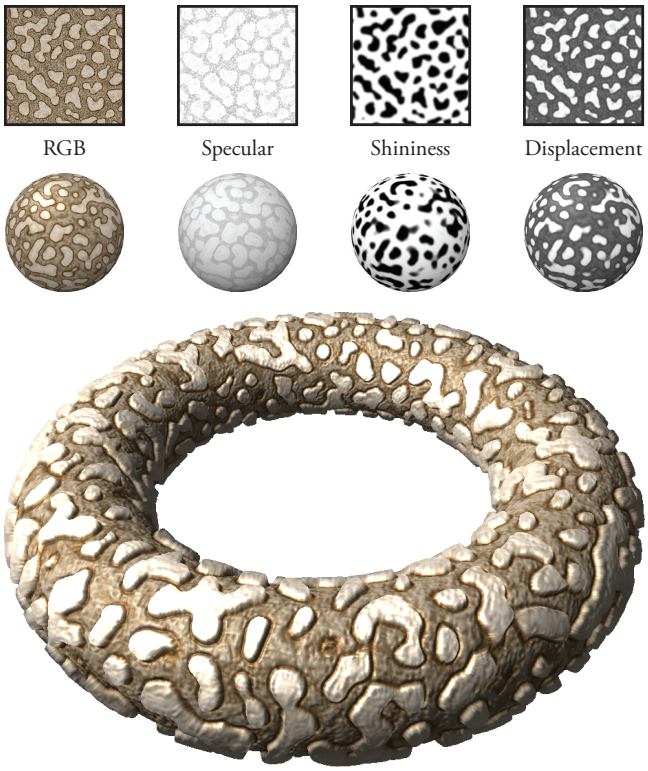

Figure 7: Synthesis of a 6-channel solid. The top row shows the channels of the 2D exemplar. The spheres show the corresponding channels from the synthesized texture. Note the correlation between features in the synthesized channels. Bottom: object rendered using all six channels in the synthesized solid.

same observations are true for the Aura 3D [Qin and Yang 2007] results shown in the figure. In addition, these results exhibit some color smearing due to the independent synthesis of correlated color channels. It should be noted that the Aura 3D paper and website contain a large number of other textures on which that method seems to perform remarkably well. However, all of these other results are generated from input exemplars which are fairly unstructured, and have decorrelated color channels to begin with, which is not that common in practice. Furthermore, we have not been able to reproduce these results. The Aura 3D results that we chose to include in our comparison are the only ones we found that use input exemplars from other texture synthesis works, and they are also the only ones with correlated color channels.

Our approach produces results comparable to those produced by stereological texture synthesis [Jagnow et al. 2004], but our method did not require modeling the particle shapes. Our method did make use of a feature map in these cases. Additional examples may be found in the supplementary materials.

6.1 Multi-channel textures

Our algorithm is directly applicable to multi-channel textures, where each exemplar texel is a high-dimensional vector. As an example, consider the exemplar shown in Figure 7 (top row). In addition to three color channels, each texel consist of three additional values: displacement, shininess, and specularity. These additional channels were crafted by hand from the original color texture, and are highly correlated with it. In such cases, independent channel synthesis is doomed, and capturing the cooccurrence probabilities

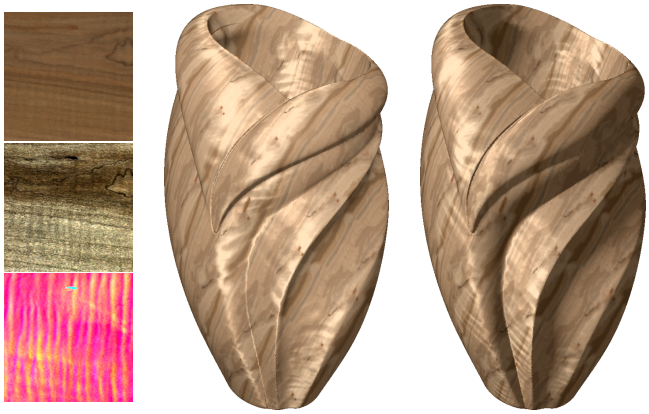

Figure 8: Finished curly maple texture. Left: the 9-channel exemplar (top: diffuse reflectance channels, middle: fiber reflectance channels, bottom: fiber axis channels). Middle and right: two renderings with different lighting of a vase textured using the resulting 9-channel solid. The renderings correctly exhibit the shifting anisotropic highlights characteristic of this type of finished wood (see supplemental video).

(as in [Qin and Yang 2007]) of the 6D texels is simply impractical for the reasons explained in Section 2.

With our method we were able to synthesize a 128^3 volume of 6D voxels, and the results (rendered with displacement mapping on programmable hardware) are shown in Figure 7. Using a PCA projection in this example reduces the dimensionality of a neighborhood vector from 384 to 19. Because the channels are correlated, we found the reduced neighborhood size (and hence the synthesis time) scale sublinearly with the number of channels.

As another example, we were able to synthesize a solid texture from one of the finished wood textures of Marschner *et al.* [2005]. These textures have nine channels per texel, containing parameters for a sophisticated RenderMan shader that simulates the appearance of finished wood. Figure 8 shows an object textured using the resulting nine-channel solid. As may be seen in the figure, the first six channels of the exemplar are strongly correlated. The solid synthesized by our method may be used to render convincing images of objects made from this type of wood, as demonstrated in Figure 8 and in the supplemental video.

6.2 Synthesis Control

Our approach offers the user several simple means of controlling the outcome of the synthesis process. First, similarly to several previous approaches [Dischler et al. 1998; Wei 2002; Qin and Yang 2007], a different exemplar may be specified for each of the orthogonal "views" of the volume. Thus, the optimization process attempts to match each of the three orthogonal neighborhoods around each voxel to one on a different exemplar.

An example is shown in Figure 9. The solid on the left was synthesized with all three views constrained to match the zebra pattern shown next to the solid. The solid on the right was generated with a polka dot pattern as the exemplar for the top view. The result is a very different solid, where horizontal slices look similar to the polka dot pattern, while vertical slices still look similar to the zebra pattern.

Second, in several cases, we found it helpful to constrain only two, rather than three views, making is easier for the synthesis process to find a good minimum. For example, the wood texture shown in Figure 1 was generated in this manner.

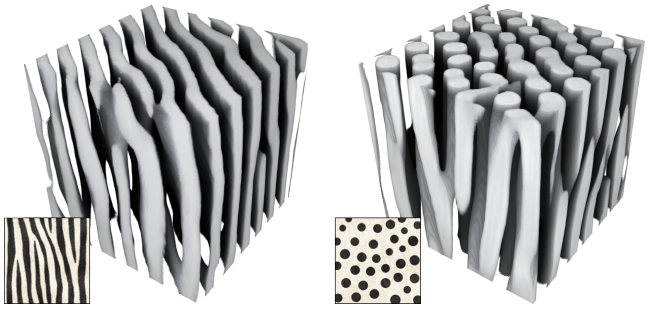

Figure 9: Synthesis control by constraining different views with different exemplars.

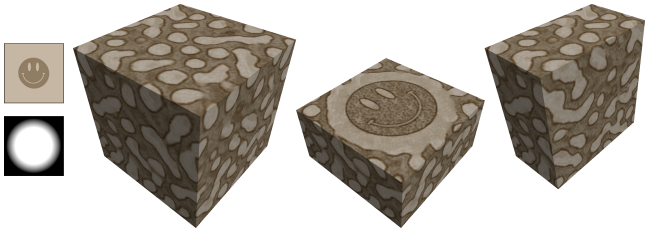

Figure 10: Constraining colors in the solid. Colors on a single slice in the solid were constraint with the maps shown on the left. Note how the other regions of the solid are unaffected, and the smiley cannot be detected in a perpendicular slice through the solid, shown on the right.

Finally, we can also directly constrain colors of specific voxels in the volume. The user specifies a volume of constraint colors, and constraint weights. During synthesis, we use the following reweighting scheme to enforce the constraints:

$$\omega'_{u,i,v} = \frac{\omega_{u,i,v}}{1 + \left\| \mathbf{e}_{u,i,v} - t_v \right\| \tau_v} \tag{6}$$

Here, t_v denotes the constraint color and τ_v the constraint weight for voxel v. The intuition is, similarly to the histogram matching, to downweight any exemplar colors that pull the synthesized solid away from the specified target. The weights τ_v allow to restrict the matching only to some areas in the solid.

Figure 10 shows an example of this kind of controllability. The image in the center shows a slice of a solid, which we have constraint with the color and weight maps shown on the left. Since other voxels were not constrained, the remaining parts of the solid are unaffected, as can be seen in the view of the whole cube or the perpendicular slice.

6.3 Limitations

While our method is able to extend a surprisingly wide variety of 2D exemplars to solids (sometimes it was not even clear to us in advance what such an extension should look like), there are some cases where it fails to synthesize coherent structures.

In many of these cases we are still able to successfully synthesize a solid texture that preserves such structures by providing a feature map as described in Section 6. However, there are cases where, even with a feature map, we were not able to generate a solid texture that would meet our expectations. Figure 11 shows such a case. The 2D exemplar here is a stone texture that exhibits a cellular structure, but the solid extension of this exemplar contains many elongated tube-like cells. Thus, when slicing the volume in certain directions,

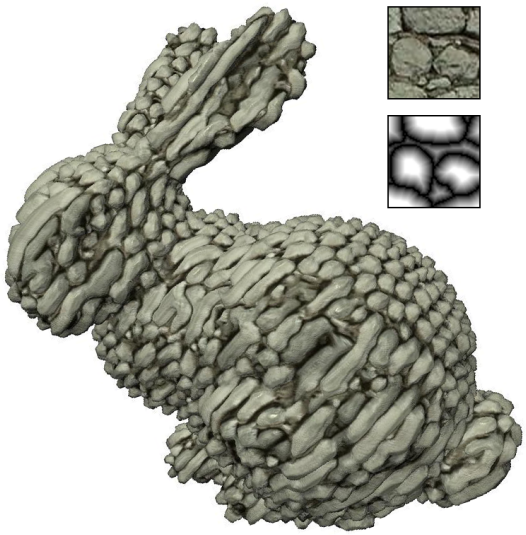

Figure 11: A failure case. Our method was not able to preserve the cellular characteristics of the exemplar, but produced elongated features instead.

the result looks quite different from the 2D exemplar, which is not what the user might expect. A possible solution to this problem might be to include additional slices in the optimization. This is an interesting direction for future work.

Finally, even with the Mean-Shift clustering technique described in Section 4.1, some fine grain detail may be lost due to smoothing. This may be seen, for example, in our comparison with the results of Jagnow *et al.* [2004] in Figure 6.

7 Conclusion

We have presented a new method for the challenging task of synthesizing solid textures from 2D exemplars. We have successfully extended 2D texture optimization techniques to the task, and integrated them with global histogram matching. Our results demonstrate that our new method is applicable to a wide variety of textures, including anisotropic textures, textures with large coherent structures, and multi-channel textures. This wide applicability and the quality of our results significantly exceeds that of previous automatic methods. We believe that our method brings example-based texture synthesis to the point where it is truly a practical and useful tool for image synthesis. Furthermore, we are excited about the possibilities of using example-based texture synthesis as a tool for 3D modeling.

Interesting directions for future research include further improving the quality and the speed of the synthesis. For example, we plan to experiment with additional kinds of histograms, in addition or instead the per-channel histograms we use right now. Another way to improve some textures might be to include additional slices in the optimization. We also plan to extend our method to synthesize a set of Wang cubes, instead of a single cube with toroidal boundary conditions. This should enable generation of high resolution solid textures, while eliminating visible periodicities and avoiding excessive storage requirements. Another promising and important direction is to develop additional control mechanisms that would enable more control via simple and intuitive interfaces.

Acknowledgements

We would like to thank Man-Kang Leung for his help in the shader development (supported by the HKSAR RGC Earmarked Grant: Project No. HKUST612505). This research was supported in parts by grants from the the following funding agencies: the Lion foundation, the GIF foundation, the Israel Science Foundation, and by DFG Graduiertenkolleg/1042 "Explorative Analysis and Visualization of Large Information Spaces" at University of Konstanz, Germany. The "fertility" model in Figure 1 was taken from the AIM@SHAPE shape repository.

References

BAR-JOSEPH, Z., EL-YANIV, R., LISCHINSKI, D., AND WERMAN, M. 2001. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics 7*, 2, 120–135.

CHEN, Y., TONG, X., WANG, J., LIN, S., GUO, B., AND SHUM, H.-Y. 2004. Shell texture functions. *ACM Transactions of Graphics 23*, 3 (Proc. SIGGRAPH 2004), 343–353.

DE BONET, J. S. 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. *Proceedings of SIGGRAPH '97*, 361–368.

DISCHLER, J.-M., GHAZANFARPOUR, D., AND FREYDIER, R. 1998. Anisotropic solid texture synthesis using orthogonal 2D views. *Computer Graphics Forum 17*, 3 (Proc. Eurographics 1998), 87–96.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, 341–346.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. *Proceedings of ICCV '99 2*, 1033–1038.

ELFADEL, I. M., AND PICARD, R. W. 1994. Gibbs random fields, cooccurrences, and texture modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*, 1, 24–37.

GHAZANFARPOUR, D., AND DISCHLER, J.-M. 1995. Spectral analysis for automatic 3-D texture generation. *Computers and Graphics 19*, 3, 413–422.

GHAZANFARPOUR, D., AND DISCHLER, J.-M. 1996. Generation of 3D texture using multiple 2D models analysis. *Computer Graphics Forum 15*, 3 (Proc. Eurographics 2006), 311–323.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. *Proceedings of SIGGRAPH '95*, 229–238.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. *Proceedings of SIGGRAPH 2001*, 327–340.

JAGNOW, R., DORSEY, J., AND RUSHMEIER, H. 2004. Stereological techniques for solid textures. *ACM Transactions on Graphics 23*, 3 (Proc. SIGGRAPH 2004), 329–335.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics 22*, 3 (Proc. SIGGRAPH 2003), 277–286.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Transactions on Graphics 24*, 3 (Proc. SIGGRAPH 2005), 795–802.

LEFEBVRE, S., AND HOPPE, H. 2006. Appearance-space texture synthesis. *ACM Transactions on Graphics 25*, 3 (Proc. SIGGRAPH 2006), 541–548.

LEFEBVRE, L., AND POULIN, P. 2000. Analysis and synthesis of structural textures. *Proceedings of Graphics Interface 2000*, 77–86.

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *Proceedings of SIGGRAPH 2001*, 127–150.

LIU, X., HU, Y., ZHANG, J., TONG, X., GUO, B., AND SHUM, H.-Y. 2004. Synthesis and rendering of bidirectional texture functions on arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics 10*, 3, 278–289.

MARSCHNER, S. R., WESTIN, S. H., ARBREE, A., AND MOON, J. T. 2005. Measuring and modeling the appearance of finished wood. *ACM Transactions on Graphics 24*, 3 (Proc. SIGGRAPH 2005), 727–734.

MOUNT, D. M., AND ARYA, S., 2006. ANN: A library for approximate nearest neighbor searching.

PEACHEY, D. R. 1985. Solid texturing of complex surfaces. *Proceedings of SIGGRAPH '85*, 279–286.

PERLIN, K. 1985. An image synthesizer. *Proceedings of SIGGRAPH '85*, 287–296.

QIN, X., AND YANG, Y.-H. 2002. Estimating parameters for procedural texturing by genetic algorithms. *Graphical Models 64*, 1, 19–39.

QIN, X., AND YANG, Y.-H. 2007. Aura 3D textures. *IEEE Transactions on Visualization and Computer Graphics 13*, 2, 379–389.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. *Proceedings of SIGGRAPH 2000*, 489–498.

SOATTO, S., DORETTO, G., AND WU, Y. 2001. Dynamic textures. *Proceedings of CVPR 2001 II*, 439–446.

SZUMMER, M., AND PICARD, R. 1996. Temporal texture modeling. *Proceedings of IEEE conference on image processing '96*, 823–826.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics 21*, 3 (Proc. SIGGRAPH 2002), 665–672.

TURK, G. 1991. Generating textures on arbitrary surfaces using reaction-diffusion. *Proceedings of SIGGRAPH '91*, 289–298.

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

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

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

WEI, L.-Y. 2002. *Texture synthesis by fixed neighborhood searching*. PhD thesis, Stanford University.

WEI, L.-Y. 2003. Texture synthesis from multiple sources. *SIGGRAPH 2003 Sketch*, 1.

WEXLER, Y., SHECHTMAN, E., AND IRANI, M. 2004. Space-time video completion. *Proceedings of CVPR 2004 1*, 120–127.

WEXLER, Y., SHECHTMAN, E., AND IRANI, M. 2007. Space-time completion of video. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 3, 463–476.

WU, Q., AND YU, Y. 2004. Feature matching and deformation for texture synthesis. *ACM Transactions on Graphics 23*, 3 (Proc. SIGGRAPH 2004), 364–367.

YING, L., HERTZMANN, A., BIERMANN, H., AND ZORIN, D. 2001. Texture and shape synthesis on surfaces. *Rendering Techniques (Proc. Eurographics Workshop on Rendering 2001)*, 301–312.