

Aura 3D Textures

Xuejie Qin and Yee-Hong Yang
Computer Graphics Lab
Department of Computing Science
University of Alberta
Edmonton, Canada
{xuq, yang}@cs.ualberta.ca

Abstract

This paper presents a new technique, called *aura 3D textures*, for generating solid textures based on input examples. Our method is fully automatic and requires no user interactions in the process. Given an input texture sample, our method first creates its aura matrix representations and then generates a solid texture by sampling the aura matrices of the input sample constrained in multiple view directions. Once the solid texture is generated, any given object can be textured by the solid texture. We evaluate the results of our method based on extensive user studies. Based on the evaluation results using human subjects, we conclude that our algorithm can generate faithful results of both stochastic and structural textures with an average successful rate of 76.4%. Our experimental results also show that the new method outperforms Wei & Levoy’s method and is comparable to that proposed by Jagnow, Dorsey, and Rushmeier.

Keywords: Texture synthesis, Solid textures, Aura matrices, BGLAMs (Basic Gray Level Aura Matrices).

1 Introduction

In computer graphics and computer games, texture synthesis has been widely recognized as an important tool in generating realistic textures for rendering complex graphic scenes. Recent advances in 2D texture synthesis [1, 11, 12, 18, 22, 25, 41] have ignited the development of many successful techniques for generating surface textures from input samples [3, 9, 26, 38, 42, 46]. Although a wide range of textures can be synthesized in 2D, there is still a lack of techniques in generating 3D textures. When 2D textures are used in texturing 3D objects, the following disadvantages are found: (1) the distortion problem on large-curvature

surfaces, and (2) non-reusable – textures generated for one surface cannot be used for other surfaces. The second limitation makes 2D surface textures difficult, if not impossible, to be used in procedural shaders [10].

To overcome the above problems, solid textures [29, 30] can be used. A solid texture is considered as a block of colored points in 3D space to represent a real-world material, for example, a wood trunk. Once the solid texture is available, any given 3D object can be textured by carving the object out of the volumetric data. Since solid textures define colors for each point in 3D space, they avoid the problems of distortion and discontinuity. However, solid textures are far more difficult to obtain than 2D textures; there is no easy way to obtain solid textures from real-world materials. Over the last two decades, procedural techniques and image-based techniques have been developed to generate solid textures. In procedural approaches [10], procedures are designed and called to generate solid textures with the surface appearance of realistic objects, such as wood, stone, smoke, fire, fluid, cloud, etc. However, these techniques can model only a limited range of textures. In addition, the procedures are difficult to understand and control because there are many parameters in the procedures and these parameters are not intuitive for a user to determine their appropriate values. To address these problems, a number of researchers have developed image-based techniques [7, 8, 18, 20, 24, 40] for synthesizing solid textures from input samples, and appealing results have been obtained. Unfortunately, some of these techniques are not fully automatic and involve nontrivial user interactions [7, 20]; while others may apply to only limited types of textures [8, 18, 20, 24, 40].

In this paper, we present a new method, called *aura 3D textures*, for generating solid textures from input samples automatically without user interactions. In theory, our method can take any number of input samples. As shown in Figure 1, given one or more input textures, our method first characterizes each input sample as a set of aura matrices [34]. Once the aura matrices are calculated, the input will not be needed. A solid texture is generated by sampling the aura matrices

of the inputs. The details of the aura 3D sampling are described in Section 4.2. After the solid texture is obtained, any given object can be textured by the solid texture using a shader.

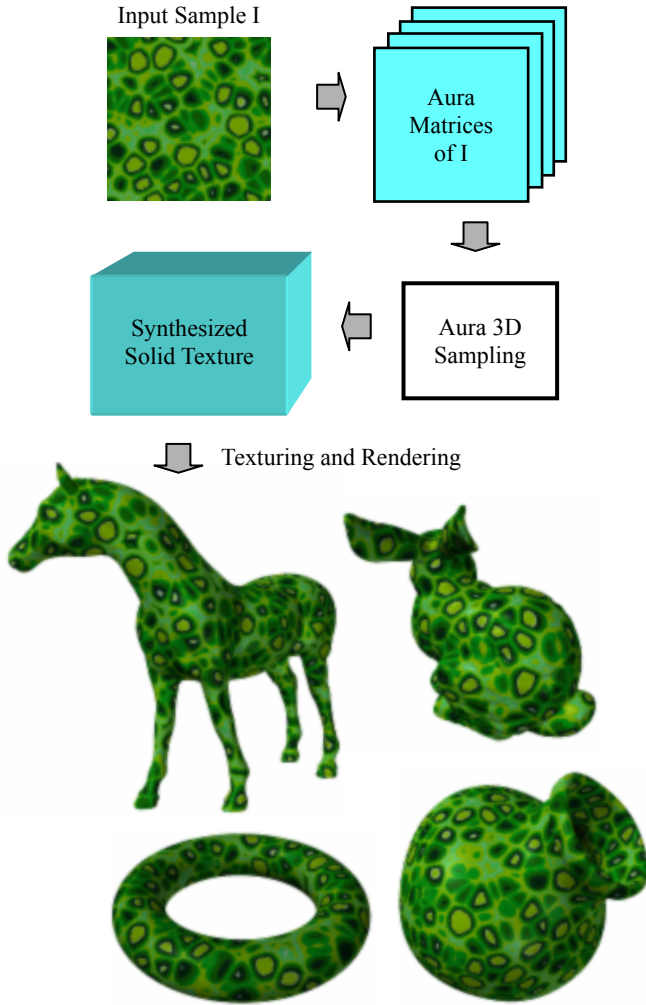


Figure 1: An overview of aura 3D textures.

We have compared our algorithm with two recently proposed algorithms: Wei & Levoy’s [40]; and Jagnow, Dorsey, and Rushmeier’s [20]. The experimental results show that our method outperforms Wei & Levoy’s and is comparable to that of Jagnow et al.’s. However, the latter method involves extensive user interactions in designing appropriate 3D shapes as well as in estimating the correct cross sectional profile; while our method is fully automatic with no user interactions in generating solid textures. In addition, their method can take a single input only; while ours can generate solid textures from multiple inputs.

To test the accuracy of our aura 3D texture approach, we present an evaluation method based on extensive user studies in Section 7. To avoid manual paper work, we have designed a GUI-based system to collect data and to perform the evaluation efficiently. The evaluation results show that our algorithm can generate appealing results for a wide range of textures, including both stochastic and structural textures, with an average successful rate of

76.4%.

2 Related Works

In 3D texturing, there are four ways to generate synthetic textures onto 3D surfaces: texture mapping, procedural texturing, image-based surface texturing, and image-based solid texturing. Texture mapping [17] is the earliest approach to generating synthetic textures on surfaces of computer-generated objects. Since Blinn’s work [2], various techniques [19, 21, 35, 39, 45, 47] have been developed to synthesize high quality textures efficiently on 3D surfaces. In general, texture mapping suffers the well-known problems of distortion, discontinuity, and unwanted seams.

The second approach is called procedural texturing [10]. Since the seminal works of Cook [4], Peachey [29], and Perlin [30], procedural techniques have been widely accepted in the computer graphics community. In most existing techniques, storage-efficient procedures using basis functions [5, 23, 30, 43] can create high quality 3D textures with no distortion and no discontinuity. Some techniques use the reaction-diffusion processes [14, 37] to generate biological patterns, e.g. zebra stripes and cellular patterns, that are found on animal skins. The disadvantages of procedural texturing include: (1) only limited types of textures can be modeled, (2) the design of procedures is based on the experience of the designer and is largely a manual process, and (3) the parameters of a texturing procedure are difficult to tune.

The third approach is the image-based surface texturing developed by a number of researchers recently. Wei & Levoy [42], Ying et al. [44], and Turk [38] have concurrently extended Wei & Levoy’s 2D texture-synthesis algorithm [41] to synthesize textures onto arbitrary mesh surfaces. Using feature-based warping and texton masks, Zhang et al. [46] have successfully synthesized progressively-variant textures onto 3D surfaces from multiple input samples. In Chen’s work [3], shell texture functions are used to synthesize realistic textures with translucency variations on surfaces from either 2D or 3D samples, e.g. a block of CT scan. Recent research works [26, 36] have also been done in generating bidirectional texture functions (BTF) onto 3D mesh surfaces. Compared with procedural texturing, image-based surface texturing can synthesize a wide range of textures. However, the approach may still suffer the distortion problem on surfaces where the curvature is large. Another problem of the approach is that textures generated for one surface cannot be used for other surfaces. This limitation makes the techniques difficult to be used in procedural shaders [10].

To combine the advantages of the procedural texturing and the image-based 2D texture analysis and synthesis, several researchers have developed techniques for generating solid textures from input samples, which

we call image-based solid texturing. Different from image-based surface texturing, these techniques synthesize a volumetric texture data from input samples. Once the volumetric data is generated, it can be used to texture different objects. In Heeger and Bergen's work [18], homogeneous and stochastic 3D textures are successfully generated by matching the histogram of a volumetric data with that of the input sample from coarse to fine resolutions. However, their approach fails for structural textures. To address this problem, Dischler et al. [8] propose a method based on spectral and histogram analysis to synthesize a wider range of solid textures from input samples. Although only a limited range of textures can be modeled, Dischler et al.'s method [8] is the first approach capable of generating structural solid textures such as wood and marble. By analyzing and extracting parameters from input images, Lefebvre and Poulin's algorithm [24] is also able to synthesize some structural textures such as wood and regular tiles. Wei [40] and Paget [27] have extended their respective 2D texture synthesis algorithms [28, 41] to generate structural solid textures as well as stochastic textures. However, both approaches work for only a limited range of textures. More recently, in Jagnow, Dorsey and Rushmeier's work [20], a stereology-based approach is presented to generate a limited range of solid textures, in particular, marble-like textures. In their approach, in order to generate the correct results, extensive user interactions are required in creating 3D particles of the desired shapes and of the required distributions. Dischler and Ghazafarpour [7] have also developed an interactive image-based framework for synthesizing structural solid textures of certain types.

Our work belongs to the category of image-based solid texturing. In particular, we present a BGLAM-based framework for synthesizing solid textures from 2D input samples. Additionally, we describe how to evaluate the results of our method using extensive user studies based on a carefully designed GUI-based system. The new approach is motivated by our recent work on 2D texture analysis and synthesis [34]. It is most related to Heeger and Bergen's [18] and Dischler et al.'s methods [8]. However, the texture analysis process of our method is done using BGLAMs rather than using gray level histograms [18] or spectrum in the frequency domain [8] (Note: Dischler et al.'s method also uses histogram-analysis to characterize textures). In the synthesis process, our method generates solid textures by sampling only the BGLAMs of the inputs. On the other hand, Heeger and Bergen's method needs filters to build pyramids for the input and output, and the synthesis results of their method heavily depend on the selection of filters. While there is no need for filters in Dischler et al.'s approach, it cannot synthesize textures with edges [8]. Both Heeger and Bergen's and Dischler et al.'s methods fail for large structural textures such as bricks;

while our method can generate appealing results for such structural textures as shown in the paper.

3 BGLAM Concepts and Theory

Our work is based on our recently proposed BGLAM (Basic Gray Level Aura Matrices) mathematical framework [34], which is developed based on the aura concepts (i.e. aura sets, aura measures, and aura matrices) originally proposed by Elfadel and Picard [13]. Under the BGLAM framework, an image X is modeled as a finite rectangular lattice S of $m \times n$ grids with a neighborhood system $N = \{N_s, s \in S\}$, where N_s is the neighborhood at site s . The neighborhood N_s at site s can be viewed as a translation of a basic neighborhood [13], denoted E , which is called the *structuring element* for the neighborhood system N . A single site neighborhood system is a system with a structuring element that contains a single neighboring site.

Aura Set: [13] Given two subsets $A, B \subseteq S$, the aura set of A with respect to B for neighborhood system N , denoted $\mathcal{G}_B(A, N)$ (or $\mathcal{G}_B(A)$ when N is understood), is given by:

$$\mathcal{G}_B(A) = \mathcal{G}_B(A, N) = \bigcup_{s \in A} (N_s \cap B). \quad (1)$$

Aura Measure: [13] With the same notations as in Eq. 1, the aura measure of A with respect to B , denoted $m(A, B)$, is given by:

$$m(A, B) = m(A, B, N) = \sum_{s \in A} |N_s \cap B|, \quad (2)$$

where for a given subset $A \subseteq S$, $|A|$ is the total number of elements in A .

GLAM (Gray Level Aura Matrix): [13] Let N be the neighborhood system over S , and $\{S_i, 0 \leq i \leq G-1\}$ be the gray level sets of an image over S , then the GLAM of the image over N , denoted A , is given by:

$$A = A(N) = [a_{ij}] = [m(S_i, S_j)], \quad (3)$$

where G is the total number of gray levels in the image, $S_i = \{s \in S \mid x_s = i\}$ is the gray level set corresponding to the i^{th} level, and $m(S_i, S_j)$ is the aura measure between S_i and S_j given by Eq. 2, and $0 \leq i, j \leq G-1$.

BGLAM (Basic GLAM) [34]: A BGLAM is a GLAM computed from a single site neighborhood system.

The aura of A with respect to B characterizes how the subset B is present in the neighborhood of A . An example of an aura on a binary lattice with the four-nearest-neighbor neighborhood system is shown in Figure 2. The aura measure $m(A, B)$ measures the amount of B 's sites presented in the neighborhood of A . Note that $m(A, B)$ does not measure the number of elements in the aura of A w.r.t B , i.e. in general,

$m(A, B) \neq |\mathcal{G}_B(A)|$. In the example shown in Figure 2, we have $m(A, B) = 12 \neq 10 = |\mathcal{G}_B(A)|$. The GLAM of an image measures the amount of each gray level in the neighborhood of each other gray level. The GLAM for the binary image shown in Figure 2 (a) is

$$A = \begin{bmatrix} 48 & 12 \\ 12 & 8 \end{bmatrix},$$

which is calculated using the structuring element of the four-nearest-neighbor neighborhood system as shown in Figure 2 (b).

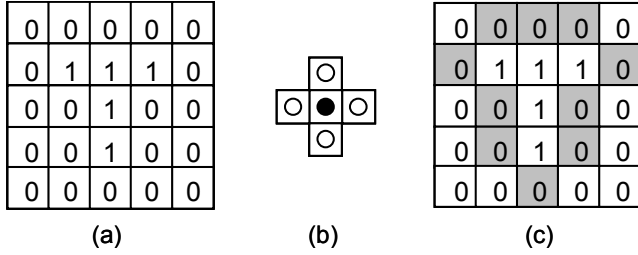


Figure 2: An example of an aura on a binary lattice with the four-nearest neighbors. (a) A sample binary lattice S , where the subset A is the set of all 1's and B the set of all 0's. (b) The structuring element of the neighborhood system. (c) The set of shaded sites is the aura set of A w.r.t to B .

The main theory on BGLAMs is presented in the following theorem. For the proof, the interested reader is referred to Qin and Yang's paper [34].

Theorem Two images are identical if and only if their corresponding BGLAMs are the same.

The above theorem implies that an image can be uniquely represented by its BGLAMs. It is noteworthy that, in the theorem, Qin and Yang [34] prove that an image can be reconstructed by its BGLAMs, but not by symmetric GLAMs [13, 31, 32], nor by GLCMs (Gray Level Cooccurrence Matrices) [6, 16, 48].

Intuitively, the BGLAMs of an image characterize the cooccurrence probability distributions of gray levels at all possible displacement configurations and thus estimate the underlying stochastic process that is used to generate a given texture sample. However, BGLAMs should not be confused with GLCMs. In fact, for 2D texture synthesis, it is shown that the method based on BGLAMs significantly outperforms the one based on GLCMs [34].

4 Aura 3D Textures

An overview of our approach is given in Figure 1. Our approach can take a single input sample or multiple input samples. Given an input texture sample, as shown in Figure 1, our method first characterizes the input so that the given sample texture can be well represented. Since a texture image can be accurately represented by and faithfully reconstructed from its *basic gray level*

aura matrices (BGLAMs) [34], we use BGLAMs to characterize and parameterize a texture sample. In aura 3D sampling, a solid texture is generated by matching the BGLAMs of volumetric data's slices with the BGLAMs of the input in multiple view directions, e.g. the positive directions of the x , y , and z -axes of the 3D coordinate system. Once the solid texture is generated, a shader can be used to texture different objects. The details of our approach are described as follows.

4.1 Calculating the Aura Matrices

In this paper, a compact set of BGLAMs defined over a neighborhood system (e.g. a 9×9 square window) is used to characterize input samples parametrically. Once the BGLAMs of an input sample are calculated, the input is no longer needed and only the BGLAMs are used in subsequent aura 3D sampling to generate the solid texture.

For an $n \times n$ neighborhood system, the total number of BGLAMs is $n^2 - 1$ because there are $n^2 - 1$ neighboring pixels around the central target pixel, and each neighboring pixel accounts for a BGLAM. An example of a 5×5 binary image and its BGLAMs calculated over a 3×3 square window are shown in Figure 3.

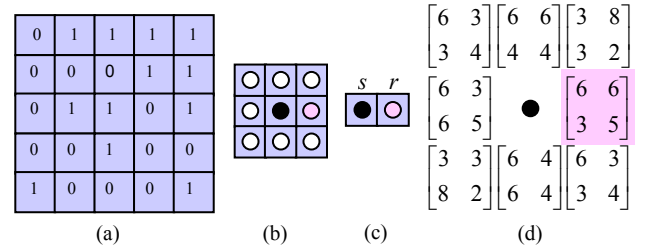


Figure 3: The BGLAMs of a 5×5 binary image. (a) The binary image, (b) the 3×3 neighborhood system, (c) the displacement configuration of neighboring pixel r , and (d) the corresponding BGLAMs of eight displacement configurations. For ease of reference, the BGLAMs in (d) are placed according to their displacement configurations in (b). For example, the BGLAM shaded in purple in (d) is for the displacement configuration of the purple pixel in (b).

To calculate the BGLAM of a specific displacement configuration, e.g. the one shown Figure 3 (c), each entry of the BGLAM $A = [a_{ij}]$ is initialized to zero, i.e.

$a_{ij} = 0$ for $0 \leq i, j \leq m - 1$, where m is the total number of gray levels in the image. For each site s in the image, let g be its gray level, and g' the gray level of its neighboring site r in the displacement configuration. Then, the value of $a_{gg'}$ is incremented by 1. After all the sites in the image have been processed, the calculation of the BGLAM is finished. When handling a

target site on the image boundaries, we consider only its neighboring sites inside the image and discard those outside of the image.

4.2 Aura 3D Sampling

For illustration purposes, we describe the aura 3D sampling in the case of three input samples. The situation for fewer or more input samples can be handled similarly. The general flow of the aura 3D sampling is given in Figure 4.

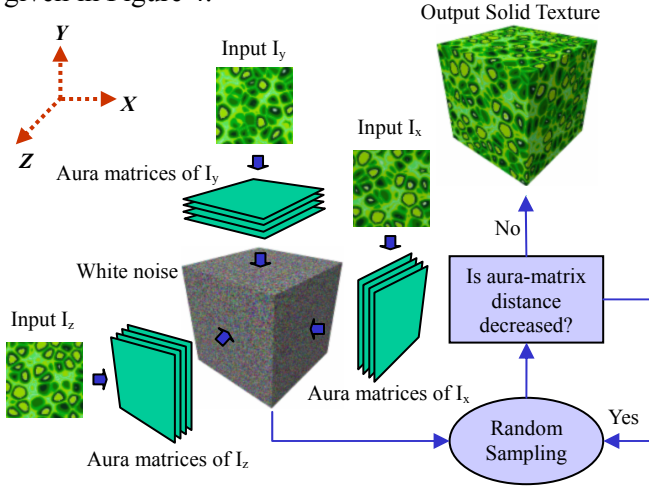


Figure 4: The general flow of aura 3D sampling.

In the beginning, the aura matrices of input samples are calculated using the algorithm described in Section 4.1 and a volume of white noise is initialized. The aura matrices of each input is used to define constraints in a specific view direction during sampling such that the final synthesized volume will have similar texture to the corresponding input sample when a cross section perpendicular to that view direction is cut from the volume. In Figure 4, for example, the aura matrices of input I_x are used to constrain the sampling in the x direction to make sure the slices of the output volume in that direction look similar to I_x . For the case of single input sample, the aura matrices constrained in a view direction is calculated either from the input or from the rotated version of the input. The view directions for adding constraints can be arbitrary in our algorithm. For example, to generate a solid texture of a regular octahedron (a polyhedron with eight equilateral triangles as faces), eight input samples can be placed along the norm directions of the octahedron's eight faces. For the purpose of illustration, the view directions in Figure 4 are demonstrated as in the positive directions of the xyz -axes.

After initialization, the algorithm iteratively modifies the noise such that the aura-matrix distance defined in Eq.4 in Section 4.3 between the xyz -slices of the volume and the input samples is decreased as much as possible. The intuition behind this is as follows: two textures are guaranteed to look similar if their corresponding

BGLAMs are close enough [34]. We use the weighted-sum distance (see Eq. 4) because we want to make sure that the points in the volume closer to a view direction are more likely synthesized by the input sample constrained in that direction and that there is a smooth transition between textures of different views. The calculation of weights, which is discussed later, depends only on the points in the volume and the view directions and thus is automatically done by the algorithm.

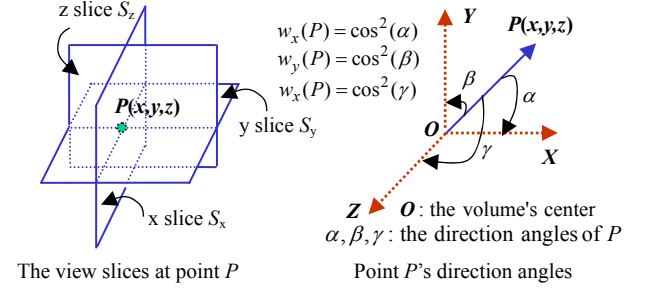


Figure 5: The view slices and the direction angles of point P in volume V .

During sampling, each point of the volume is visited *randomly* once, and its color is modified so that the distance defined in Eq.4 between the BGLAMs of the *view slices* (see the left in Figure 5) of the volume and the BGLAMs of the input samples is decreased. More precisely, when visiting a point, the algorithm first finds the candidate set of all colors different from the current color that decrease or at least do not increase the aura-matrix distance. Then, it randomly chooses a color from the candidate set as the color of the point. Note that even when the color does not change the distance, i.e. at the same distance as the current color, the algorithm still includes it into the candidate set in order to increase the randomness in the output. It is possible that the candidate set is empty at the end of search, which implies that any color different from the current color will increase the distance. In such a case, the point retains its current color and the algorithm goes to process the next point in the volume. When the distance is below a predefined threshold or when there is no change in colors at any point of the volume, the algorithm returns the volume as the final solid texture.

4.3 Weighted Aura-Matrix Distance

The aura-matrix distance used in the aura 3D sampling is defined by

$$d = d(V, I_x, I_y, I_z) = \frac{1}{|V|} \sum_{P \in V} [w_x(P) * d(S_x, I_x) + w_y(P) * d(S_y, I_y) + w_z(P) * d(S_z, I_z)] \quad (4)$$

where $|V|$ is the total number of points in volume V ; $w_x(P)$, $w_y(P)$, and $w_z(P)$ are the weights calculated

from the cosines of the *direction angles* α , β , and γ of point P in V as shown in the right in Figure 5; and $d(S_x, I_x)$ is the distance between the BGLAMs of view slice S_x and the BGLAMs of input sample I_x . Given two images X and Y , let $A(X) = \{A_i\}_{0 \leq i \leq n-1}$ and $A(Y) = \{B_i\}_{0 \leq i \leq n-1}$ be their corresponding normalized BGLAMs, then the BGLAM distance $d(X, Y)$ is given by:

$$d(X, Y) = d(A(X), A(Y)) = \frac{1}{n} \sum_{i=0}^{n-1} \|A_i - B_i\|, \quad (5)$$

where for a given aura matrix $A = [a_{ij}]_{0 \leq i, j \leq m-1}$, its norm

$\|A\|$ is given by $\|A\| = \sum_{i,j=0}^{m-1} |a_{ij}|$, and an aura matrix

$A = [a_{ij}]$ is *normalized* if $\sum_{ij} a_{ij} = 1$. Since two images

X and Y may have different sizes, the aura matrices must be normalized to make sure that there is no bias between the values of $d(S_x, I_x)$, $d(S_y, I_y)$, and $d(S_z, I_z)$ when the distance defined by Eq. 4 is calculated.

As shown in the right of Figure 5, when a point in the volume is closer to a view direction, e.g. the x -axis, there is more chance during sampling for the point to be colored by the input sample constrained in that direction. Since $\cos(\alpha)$, $\cos(\beta)$, and $\cos(\gamma)$ are continuous functions, there is a smooth transition in the synthesized textures from one view direction, e.g. the x -axis, to the other, e.g. the y -axis. For a given point $P(x, y, z)$ in the volume V , let $O(x_0, y_0, z_0)$ be the center of V , then the weights can be calculated by:

$$w_x(P) \stackrel{\text{def}}{=} \cos^2(\alpha) = \frac{(x - x_0)^2}{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}$$

$$w_y(P) \stackrel{\text{def}}{=} \cos^2(\beta) = \frac{(y - y_0)^2}{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}.$$

$$w_z(P) \stackrel{\text{def}}{=} \cos^2(\gamma) = \frac{(z - z_0)^2}{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}$$

One can verify that $w_x(P) + w_y(P) + w_z(P) = 1$. If P coincides with O , we let $w_x(P) = w_y(P) = w_z(P) = 1/3$.

4.4 Algorithm

The pseudo code of the algorithm for aura 3D textures is given in Figure 6. The definition of distance d in step 2 of the main algorithm is given in Eq. 4. In step b.2.1 of *aura3DSampling*, the view slices of V at point P are defined as shown in Figure 5.

The major computation cost of the aura 3D texture synthesis algorithm is in the two *while* loops in step 2 (see Figure 6). In an iteration of step 2 (i.e. step 2.1, which is one pass of visiting all points in V), a brute

force method, which would perform fresh recalculations each time when the aura-matrix distance is computed, has a cost of at least $O[m * np^2 * G * (S + G^2)]$, where G is number of gray levels in the input image, m the total number of BGLAMs used in the sampling, np the number of points in the volume, and S the size of the view slices of volume V . A more efficient way is to perform an incremental update based on updated information, which can be done with a cost of $O(m * np * S * G)$ because when a pixel changes its gray level value, only its neighboring pixels are affected.

Aura 3D Texture Synthesis

Input:

$I_x, I_y, I_z \leftarrow$ sample texture images.

$\varepsilon \leftarrow$ a given small positive number in $(0, 1)$

Output:

$V \leftarrow$ the synthesized solid texture.

Begin

1 Initialize V as a volume of random noise.

2 While $d = d(V, I_x, I_y, I_z) \geq \varepsilon$ do

2.1 While there are unvisited points in V , randomly choose an unvisited point P do

$\text{grayLevel}(p) \leftarrow \text{aura3DSampling}(P, d, I_x, I_y, I_z, V)$.

End of while

End of while

End of begin

aura3DSampling (P, d, I_x, I_y, I_z, V)

b.1 $C \leftarrow$ empty (the candidate set of gray levels for point P).

b.2 For each gray level $j = 0$ to $G - 1$, and $j \neq P$'s current gray level, do

b.2.1 $S_x(j), S_y(j), S_z(j) \leftarrow$ the view slices of V at point P when P has gray level j .

b.2.2 $d_j \leftarrow d(V, S_x(j), S_y(j), S_z(j))$.

b.2.3 if $d_j \leq d$, then $C = C \cup \{j\}$.

b.3 If C is empty, then $g \leftarrow$ the current gray level value of P , Else $g \leftarrow$ a randomly chosen gray level from C .

Figure 6: The pseudo code of aura 3D texture synthesis algorithm.

[**Proof of the fast-version time complexity**] Let $T(2.1)$, and $T(b)$ are the time complexity for the step 2.1 and the procedure *aura3DSampling*, respectively, then the time for one pass of step 2, is given by:

$$T(2.1) = np * T(b). \quad (6)$$

In the algorithm shown in Figure 6, the time for both step b.1 and step b.4 is a constant, and the time for step b.3 is at most $O(G)$. By using an incremental update scheme as described in Qin and Yang's paper [34], the distance d_j in step b.2.2 of the algorithm can be efficiently updated from distance d (i.e. the input distance parameter of the procedure *aura3DSampling*) with a computation cost of $O(m * S)$. The time for step b.2.1 and step b.2.3 is $O(S)$ and $O(1)$, respectively. Thus, we have the following:

$$\begin{aligned}
T(b) &= T(b.1) + T(b.2) + T(b.3) + T(b.4) \\
&\leq G * [T(b.2.1) + T(b.2.2) + T(b.2.3)] + O(G) \\
&= G * \{O(S) + O[m * S] + O(1)\} \quad . \quad (7) \\
&= G * O(m * S) \\
&= O(m * S * G)
\end{aligned}$$

By Eq. 6 and 7, we have: $T(2.1) \leq O(m * np * S * G)$. ■

SVD RGB Color Transform

Input:

$X \leftarrow$ the RGB color image.

Output:

$Y \leftarrow$ the color image with independent color channels.

$T \leftarrow$ the RGB color transformation.

$T^{-1} \leftarrow$ the inverse transformation of T .

Begin

1 Subtract the mean color from each RGB color channel.

1.1 $R, G, B \leftarrow$ the red, green, and blue channel of X , respectively.

1.2 $r, g, b \leftarrow$ the mean color of R, G , and B , respectively.

1.3 $R' \leftarrow R - r$ (subtract the mean of the red values from the red value at each pixel of R).

1.4 $G' \leftarrow G - g$.

1.5 $B' \leftarrow B - b$.

1.6 $X' \leftarrow$ the color image of R', G', B' channels.

2 Calculate the 3x3 covariance matrix C of X' .

2.1 $n \leftarrow$ the number of pixels in X' .

2.2 $D \leftarrow$ the $3 \times n$ matrix whose columns are color values of each pixel in X' .

2.3 $C \leftarrow DD^T$, where D^T is the transpose of matrix D .

3 Perform SVD on C by decomposing C into the product of three of matrices: $C = US^2U^T$, where U is orthonormal and S is diagonal.

4 Calculate the transformation T and the inverse of T .

$T \leftarrow S^{-1}U$, where S^{-1} is the inverse matrix of S .

$T^{-1} \leftarrow US$.

5 $Y \leftarrow TD$, Y is a $3 \times n$ matrix with each row representing a color channel.

6 Return Y, T and T^{-1} .

End

Figure 7: The algorithm of RGB-color transformation using SVD.

4.5 Color Input Textures

For color input textures, one cannot simply apply the basic algorithm as shown in Figure 6 to each of the RGB channels separately since the RGB components of a color image are dependent on one another. Hence, before applying the basic aura 3D texture synthesis algorithm, a color-space transformation T based on the singular value decomposition technique [33] is used to transform the R , G , and B components of an color image into three independent components R' , G' , and B' . After this decorrelation step, the basic synthesis algorithm is applied to each of the independent color components R' , G' , and B' to generate three gray-scale solid textures in the transformed color space, which are then transformed back (using the inverse transformation of T) into the RGB color space to produce the final synthesized color

solid texture. The pseudo code of the RGB color-space transformation algorithm based on SVD is given in Figure 7.

5 Acceleration

For acceleration, we extend our algorithm so that it can run texture synthesis in multiresolutions, similar to the pyramid method used in Heeger and Bergen's work [18]. However, from our experience, we find that the filtering process only complicates our algorithm. Thus we have adopted a simpler non-filter-based method, called local decimation [28], to build the multiresolution representations of the input and output. For an input color texture sample of size 64×64 with 80 BGLAMs and an output volume of size $128 \times 128 \times 128$, the average running time in a single resolution is about 10 hours on a 1.4GHz Penntium 4 PC running Windows XP Professional. With a multiresolution scheme of 4 levels and 24 BGLAMs used for each level, the running time is reduced to about 3 hours. For color images, our algorithm is further extended to synthesize the three independent color channels in parallel after the step of color-space transformation as described in Section 4.5, in which case, the running time can be further reduced to about 1 hour. Once the solid texture is generated, a given object can be textured within seconds. An average runtime of 6 seconds is recorded in our experiments.

6 Results

The window size used to calculate BGLAMs in our algorithm is an important parameter that affects the synthesis results. In general, an input texture containing large structures or favorable orientations requires a relatively large neighborhood size. For a given input texture, different synthesis results can be generated with different window sizes. Figure 8 gives an example texture and its solid textures generated by using windows of different sizes.

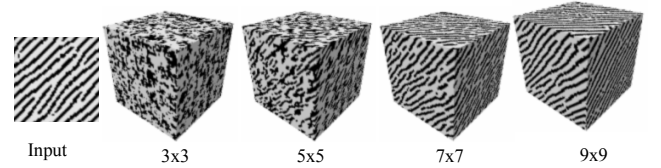


Figure 8: An example of aura 3D textures using different window sizes given under each output.

Figure 9 gives some comparison results of aura 3D textures with two existing approaches. Images in the first column are input samples of size 64×64 , images in column 2 are results of our algorithm, and images in the last column are generated by Wei & Levoy's algorithm [40] (the first three rows), and Jagnow et al.'s algorithm [20] (the last three rows), respectively. For each input texture, a solid texture of size $128 \times 128 \times 128$ is

generated using the three algorithms. In our algorithm, we use 80 BGLAMs, which are calculated over a 9×9 window, to characterize all input textures and to generate results shown in Figure 9 and in the rest of this section. As shown in Figure 9, the results of our method are better than those of Wei & Levoy’s algorithm. Compared to Jagnow et al.’s algorithm, although the result of our algorithm is not as good as theirs for the input shown in the last row, our algorithm generates better results for the inputs in row 4 and 5.

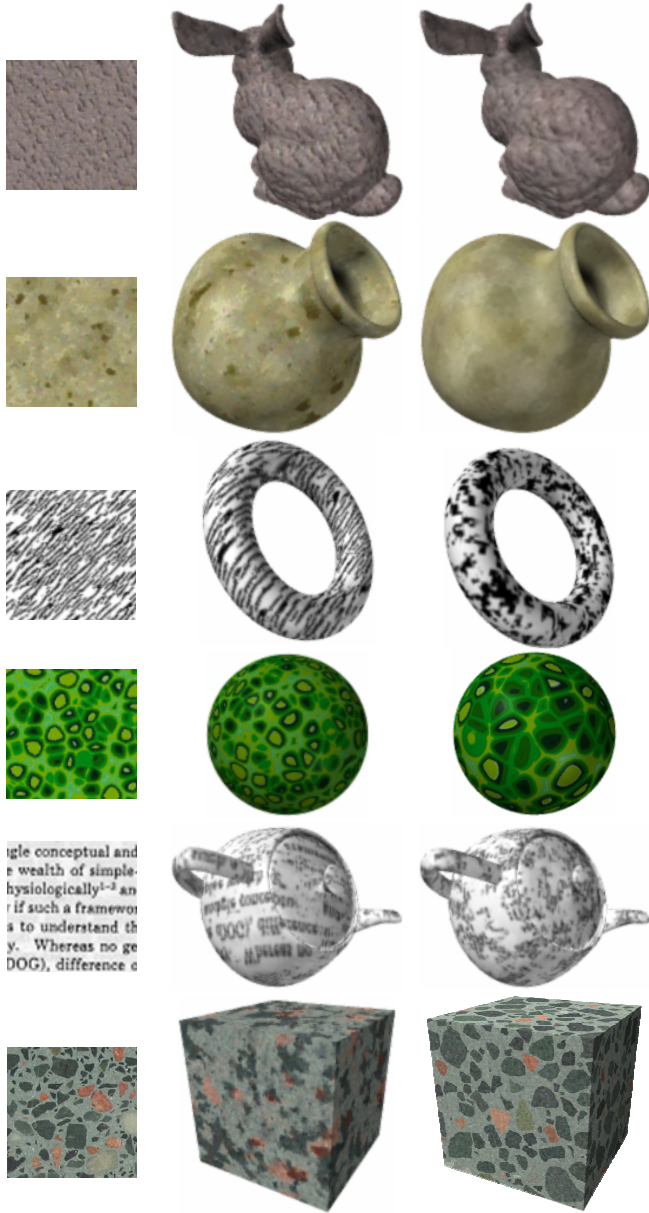


Figure 9: Comparison results of our method with Wei & Levoy’s and Jagnow et al.’s. The inputs (size 64×64) are shown in the first column, and the synthesis results are displayed in the same row as the inputs. Images in column 2 are the results of our algorithm, and images in column 3 are the results of Wei & Levoy’s (the first three rows) and Jagnow et al.’s (the last three rows).

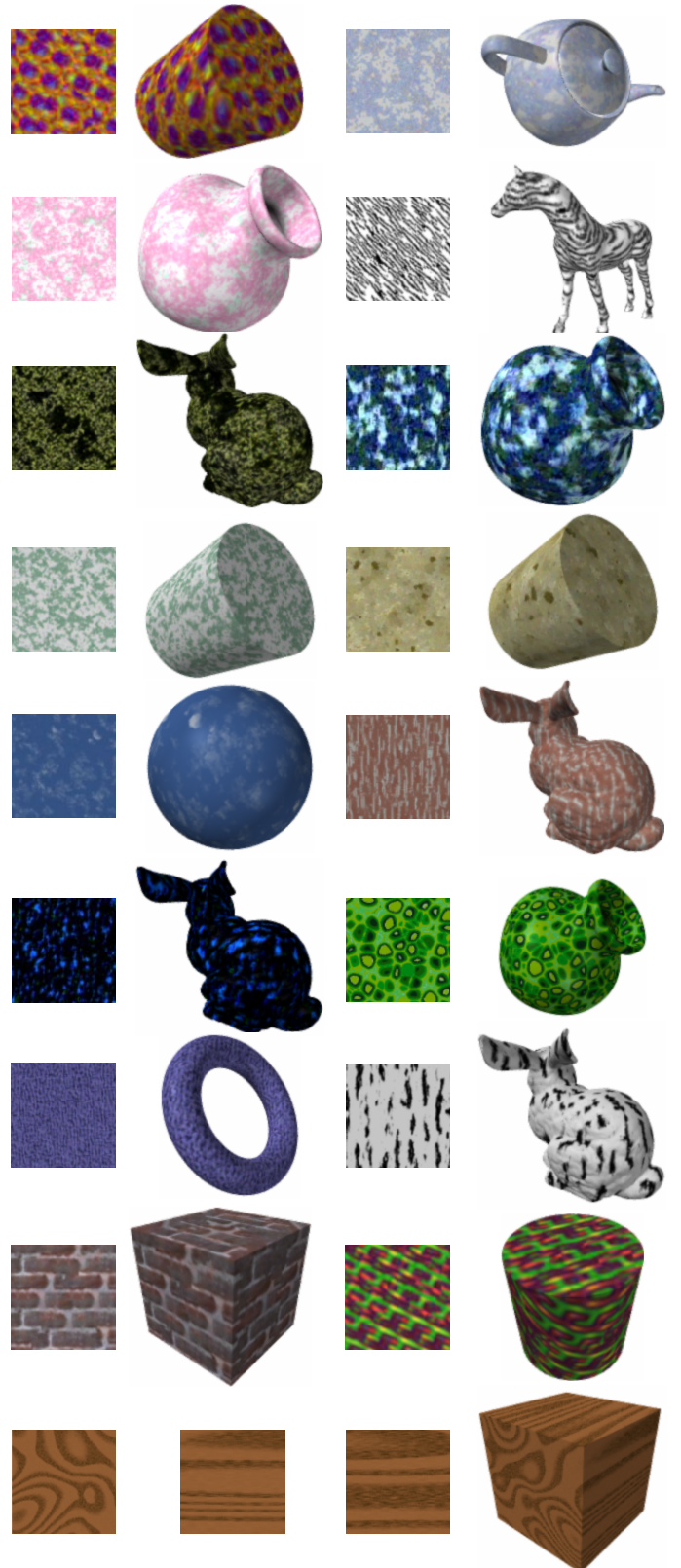


Figure 10: More results of aura 3D textures. Small patches are input samples and results of solid textures are displayed beside the corresponding inputs. In the last row, three input samples are used to generate the solid texture.

It is noteworthy that Jagnow and Dorsey’s method requires a user to manually design and to edit 3D particles to match the texture profiles of a given sample.

While it may provide flexibility to the user, it is nontrivial to design a complex texture. If the shapes of predesigned 3D particles do not match the profiles of input textures, the algorithm will generate incorrect results as the one shown in the fifth row in Figure 9. More results of our method can be found in Figure 10 and in the supplemental materials that accompany the paper.

7 Evaluation

We present a method based on user studies for evaluating our aura 3D textures results. We only describe the algorithm for the case of single input textures. It is straightforward to extend the algorithm to multiple input textures. To test the accuracy of our aura 3D textures, it is reasonable to have the following two evaluation goals: (1) we test whether or not the slices of the solid texture in each constrained direction, i.e. the direction in which the BGLAMs of the input are used to constrain the aura 3D sampling, look similar to the corresponding input; and (2) we determine whether or not textures change smoothly between consecutive slices in any view direction, including view directions that are *not* used to constrain the aura 3D sampling.

Suppose a solid texture V of size $n \times n \times n$ is synthesized from an input texture. To test the first goal, for each direction in which the BGLAMs of the input are used to constrain the sampling, we obtain all n slices of V and randomly select $m (< n)$ of them. We mix the selected m slices with other texture images that are randomly drawn from a database of texture images (a database of over 2000 images is used in our experiments), and display them in random order on the screen. Each display is evaluated by 18 people. Among them, 6 are researchers in the same research lab as the authors and have the knowledge in texture analysis and synthesis; the other 6 are graduate students in the same department and have some general knowledge in computer vision and image processing, and the remaining 6 are from outside of the department but in the same university and are in completely unrelated disciplines.

Each subject is asked to select all texture images that look similar to a specified input image. If all m slices in each of the constrained directions are selected, we consider the solid texture as a SUCCESS in terms of subject’s evaluation. If more than 50% of the subjects give an evaluation of SUCCESS on a solid texture, the solid texture is considered as a success for the first goal. Otherwise, it is a failure.

For the second goal, we randomly select v view directions (6 are used in our experiments). For each view direction, we generate an animation of all cross sections of the solid texture that are cut in order along that direction. A subject is asked to watch the animation to

determine if the texture is changing smoothly from frame to frame. This test is repeated for each of 18 people. If no sudden change has been identified in all the animations of the selected views, we consider the synthesized solid texture as a SUCCESS by the subject. If more than 50% of the subjects assign a SUCCESS to a solid texture, the solid texture is considered as a success for the second goal. Otherwise, it is a failure. Figure 11 shows example sequences of animation frames of two solid textures that are generated by our method, one for a cloud-like solid texture and another for a green-marble-like solid texture. More animations can be found in the supplemental materials that accompany the paper.

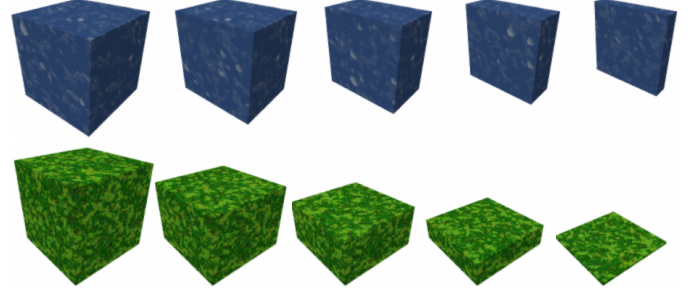


Figure 11: Animation sequences of cross sections of two solid textures that are generated by our algorithm. The texture in the first row is a cloud-like solid texture, and the one in the second row is a green-marble-like solid texture.

To avoid the manual data collection process and to make our evaluation efficient, we have designed a GUI-based evaluation system. Figure 12 gives a screen shot of the system when it is used for evaluating the first goal. The input is displayed as a smaller image in the top-left corner of the window. The m slices together with other texture images randomly selected by the system from two texture databases (one for selecting the m slices and one for selecting the other textures) are displayed on the screen in random order as larger images below the input. Texture images similar to the input are selected by just clicking the button labeled “Similar” under them. When a user finishes the test, the system will calculate and store the evaluation result. The user does not know the score until after the experiment is complete.

It is noteworthy that the user does not have any information on the value of m because otherwise he/she will use the information to guide his/her selection, which will cause a biased evaluation. In fact, in our implementation, the value of m is assigned randomly each time by the system to make sure that the user will not guess it. Furthermore, we found that the color of a texture provides an important cue in user’s selection that causes bias. For example, if color images were displayed in Figure 12, the user could easily make selections by following the color instead of the textures of the input. To avoid this bias, we converted all color images into gray scale images when testing the first goal. For testing

the second goal, we use color images because there is no such color-bias problem.

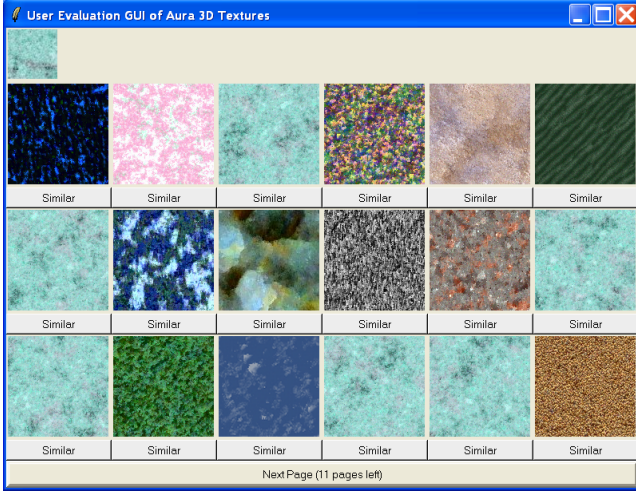


Figure 12: The GUI-based user evaluation system of aura 3D textures.

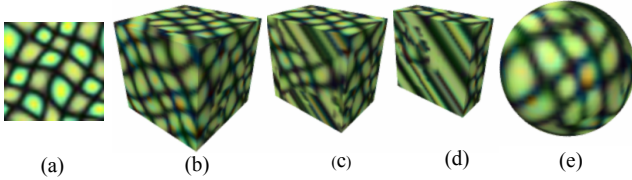


Figure 13: An example of failure from our algorithm that is identified during the evaluation. The input is shown in (a), the solid texture and its two cross sections are shown in (b), (c) and (d), respectively. A textured sphere by the solid texture is shown in (e). The failure of the solid texture in (b) is identified by viewing its cross sections as shown in (c) and (d).

For the evaluation of both goals, we have used 126 input textures, which include stochastic and structural textures. The experimental results show that the average percentage of success (for the experiments of both goals) for our aura 3D textures is 76.4%. The results of the three different groups of subjects are, respectively, 76.8%, 77.2%, and 75.1%. Thus, there is no significant difference between the results of the different groups.

Figure 13 gives an example of failure of our algorithm that is identified during evaluation. This example also demonstrates the importance of the evaluation process as discussed below. In the figure, the input and the generated solid texture of the input are shown in (a) and (b), respectively. The two cross sections of the solid texture are shown in (c) and (d). A textured sphere by the solid texture is shown in (e). By just looking at the solid texture and the textured sphere, the results look reasonably good. However, by evaluation, we have actually identified the problems of the solid texture as shown in (c) and (d), which do not appear in the textured sphere.

8 Limitations and Future Work

One limitation of the current implementation of the aura 3D texture is the color update scheme during sampling as described in Section 4.2. It is quite possible that after a few iterations, the number of candidates of possible colors for a target point is less than 3, which may sometimes cause the color values for points in the output volume to quickly converge to local minima and thus generate visible seams in the output textures as shown in Figure 13.

Future research should be carried out to address this problem. One possible solution is to extend the current single-point search scheme to a multiple-point search scheme during sampling so that the convergence to the local minima can be avoided as much as possible. Since genetic algorithms [15] are well suited for searching in multiple directions, we are currently considering them to address this local minima problem in our algorithm.

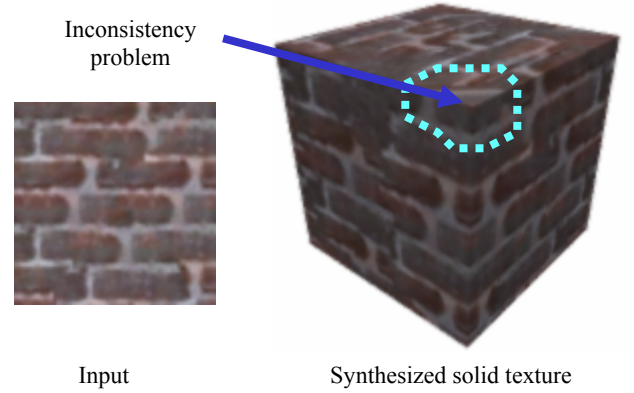


Figure 14: An example of inconsistency problems in oriented structural solid textures.

The second limitation is that although our method can generate faithful results for oriented structural textures, such as the ones shown in the second last row in Figure 10. However, we find one problem related to the issues of constraints, which we still have not found a satisfactory solution. It is relatively easy to see that the orientations of three adjacent structural textures at a junction may create an inconsistency problem. An example of such problems is shown in Figure 14, where the input sample is shown in the left and the corresponding synthesized solid texture generated by our method is shown in the right. Although the orientation of the brick at the solid's corner highlighted by the dashed line is consistent with the orientation of the bricks on both sides of the solid, it is inconsistency with that of the bricks at the top.

The solution to the above inconsistency problems in oriented structural textures, if it exists, depends on the interpretation of the given surface textures, which is a very interesting inverse problem for future research. For

example, can we detect inconsistencies? If the textures are consistent, is the solution unique?

The third limitation is related to the current implementation of the aura 3D sampling. Although our method is general enough to handle the situation in which input samples can be placed along non-orthogonal view directions to constrain the aura 3D sampling, our current implementation only handles orthogonal view directions in 3D space. We are currently considering a new implementation of our algorithm for handling non-orthogonal-view constraints.

Other future research may include: the extension of our approach for GPU-based texture synthesis and the evaluation of solid textures generated by other approaches, e.g. Jagnow et al.'s algorithm [20].

9 Conclusions

In this paper, a new method for generating solid textures from input samples is presented. Given one or more input textures, the BGLAMs of input samples are calculated first; a solid texture is then generated by sampling the BGLAMs of input samples. We evaluate the results of our method based on extensive user studies. The evaluation results show that our algorithm can generate faithful results over a wide range of textures with an average successful rate of 76.4%. The synthesized results show that our algorithm outperforms Wei & Levoy's and are comparable to Jagnow et al.'s. However, the latter method involves extensive user interactions in designing correct 3D shapes while our method is fully automatic with no user interactions in generating solid textures.

Acknowledgments

The authors would like to thank financial supports provided by NSERC, the University of Alberta, and the Karl A Clark Memorial Scholarship from the Alberta Research Council.

References

1. Ashikhmin, M., *Synthesizing Natural Textures*. The ACM Symposium on Interactive 3D Graphics, 2001: p. 217-226.
2. Blinn, J.F. and M.E. Newell, *Texture and Reflection in Computer Generated Images*. Commun. of the ACM, 1976. **19**(10): p. 542-547.
3. Chen, Y., et al., *Shell Texture Functions*. Siggraph, 2004: p. 343-353.
4. Cook, R.L., *Shade Trees*. Siggraph, 1984. **18**: p. 223-231.
5. Cook, R.L. and T. DeRose, *Wavelet Noise*. Siggraph, 2005: p. 803-811.
6. Davis, L.S., S.A. Johns, and J.K. Aggarwal, *Texture Analysis Using Generalized Cooccurrence Matrices*. IEEE PAMI, 1979. **1**(3): p. 251-259.
7. Dischler, J.-M. and D. Ghazafarpour, *Interactive Image Based Modeling of Macrostructured Textures*. IEEE Computer Graphics and Application, 1999. **19**(1): p. 66-74.
8. Dischler, J.-M., D. Ghazanfarpour, and R. Freydier, *Anisotropic Solid Texture Synthesis Using Orthogonal 2D Views*. Special issue of Proceedings of Eurographics, 1998. **17**(3): p. 87-96.
9. Dischler, J.-M., et al., *Texture Particles*. Special Issue of Proceedings of Eurographics, 2002.
10. Ebert, D.S., et al., *Texturing & Modeling: A Procedural Approach (3rd Edition)*. 2002, Academic Press.
11. Efros, A. and T. Leung, *Texture Synthesis by Non-Parametric Sampling*. IEEE ICCV, 1999: p. 1033-1038.
12. Efros, A.A. and W.T. Freeman, *Image Quilting for Texture Synthesis and Transfer*. Siggraph, 2001: p. 341-346.
13. Elfadel, I.M. and R.W. Picard, *Gibbs Random Fields, Cooccurrences, and Texture Modeling*. PAMI, 1994: p. 24-37.
14. Fleischer, K.W., et al., *Cellular Texture Generation*. Siggraph, 1995: p. 239-248.
15. Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, 1989.
16. Haralick, R.M., *Statistical and Structural Approaches to Texture*. In Proc. 4th Int. Joint Conf. Pattern Recognition, 1978: p. 45-69.
17. Heckbert, P.S., *Fundamentals of Texture Mapping and Image Warping*. Master's Thesis, in Dept. of Elec. Eng. and Compt. Sci. 1989, Univ. of California: Berkeley.
18. Heeger, D.J. and J.R. Bergen, *Pyramid-Based Texture Analysis Synthesis*. Siggraph, 1995: p. 229-238.
19. Jagnow, R. and J. Dorsey, *Virtual Sculpting with Haptic Displacement Maps*. Proceedings of Graphics Interface, 2002: p. 125-132.
20. Jagnow, R., J. Dorsey, and H. Rushmeier, *Stereological Techniques for Solid Textures*. Siggraph, 2004. **23**(3): p. 329-335.
21. Kraevoy, V., A. Sheffer, and C. Gotsman, *Matchmaker: Constructing Constrained Texture Maps*. Siggraph, 2003. **22**(3): p. 326-333.
22. Kwatra, V., et al., *Graphcut Textures: Image and Video Synthesis using Graph Cuts*. Siggraph, 2003. **22**(3): p. 227-286.
23. Lagae, A. and P. Dutré, *A Procedural Object Distribution Function*. ACM TOG, 2005. **24**(4): p. 1442-1461.
24. Lefebvre, L. and P. Poulin, *Analysis and Synthesis of Structural Textures*. Graphics Interface, 2000: p.

- 77-86.
25. Liang, L., et al., *Real-Time Texture Synthesis by Patch-Based Sampling*. ACM TOG, 2001: p. 127-150.
 26. Liu, X., et al., *Synthesis and Rendering of Bidirectional Texture Functions on Arbitrary Surfaces*. IEEE TVCG, 2004. **10**(3): p. 278-289.
 27. Paget, R., *An automatic 3D texturing framework*. Proceedings of the Intl. Workshop on Texture Analysis and Synthesis, 2005: p. 1-6.
 28. Paget, R. and I.D. Longstaff, *Texture Synthesis via a Noncausal Nonparametric Multiscale Markov Random Field*. IEEE TIP, 1998. **7**(6): p. 925-931.
 29. Peachey, D.R., *Solid Texturing of Complex Surfaces*. Siggraph, 1985. **19**(3): p. 279-286.
 30. Perlin, K., *An Image Synthesizer*. Siggraph, 1985. **19**(3): p. 287-296.
 31. Picard, R.W. and I.M. Elfadel, *Structure of Aura and Co-occurrence Matrices for the Gibbs Texture Model*. J. of Math. Imaging & Vision, 1992: p. 5-25.
 32. Picard, R.W., I.M. Elfadel, and A.P. Pentland, *Markov/Gibbs Texture Modeling: Aura Matrices and Temperature Effects*. IEEE CVPR, 1991: p. 371-377.
 33. Press, W.H., et al., *Numerical Recipes in C++: The Art of Scientific Computing (2nd ed)*. Cambridge University Press, 2002.
 34. Qin, X. and Y.H. Yang, *Basic Gray Level Aura Matrices: Theory and its Application to Texture Synthesis*. IEEE ICCV, 2005: p. 128-135.
 35. Soler, C., M. Cani, and A. Angelidis, *Hierarchical Pattern Mapping*. Siggraph, 2002. **21**(3): p. 673-680.
 36. Tong, X., et al., *Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces*. Siggraph, 2002. **21**(3): p. 665-672.
 37. Turk, G., *Generating Textures for Arbitrary Surfaces Using Reaction-Diffusion*. Siggraph, 1991. **25**(3): p. 289-298.
 38. Turk, G., *Texture Synthesis on Surfaces*. Siggraph, 2001: p. 347-354.
 39. Wang, L., et al., *View-Dependent Displacement Mapping*. Siggraph, 2003. **22**(3): p. 334-339.
 40. Wei, L., *Texture Synthesis from Multiple Sources*. Siggraph Sketches and Applications, 2003.
 41. Wei, L. and M. Levoy, *Fast Texture Synthesis Using Tree-Structured Vector Quantization*. Siggraph, 2000: p. 479-488.
 42. Wei, L.Y. and M. Levoy, *Texture Synthesis over Arbitrary Manifold Surfaces*. Siggraph, 2001: p. 355-360.
 43. Worley, S., *Cellular Texture Basis Function*. Siggraph, 1996: p. 291-294.
 44. Ying, L., et al., *Texture and Shape Synthesis on Surfaces*. Eurographics Workshop on Rendering, 2001: p. 301-312.
 45. Zelinka, S. and M. Garland, *Interactive Texture Synthesis on Surfaces Using Jump Maps*. Eurographics Symp. on Rendering, 2003: p. 90-96.
 46. Zhang, J., et al., *Synthesis of Progressively-Variant Textures on Arbitrary Surfaces*. Siggraph, 2003. **22**(3): p. 295-302.
 47. Zhou, K., et al., *TextureMontage: Seamless Texturing of Arbitrary Surfaces From Multiple Images*. Siggraph, 2005: p. 1148-1155.
 48. Zucker, S.W., *Finding Structure in Co-occurrence Matrices for Texture Analysis*. CVGIP, 1980. **12**: p. 286-308.