Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces

Xin Tong Jingdan Zhang‡ Ligang Liu Xi Wang‡ Baining Guo Heung-Yeung Shum

Microsoft Research Asia* ‡Tsinghua University†

Abstract

The bidirectional texture function (BTF) is a 6D function that can describe textures arising from both spatially-variant surface reflectance and surface mesostructures. In this paper, we present an algorithm for synthesizing the BTF on an arbitrary surface from a sample BTF. A main challenge in surface BTF synthesis is the requirement of a consistent mesostructure on the surface, and to achieve that we must handle the large amount of data in a BTF sample. Our algorithm performs BTF synthesis based on *surface textons*, which extract essential information from the sample BTF to facilitate the synthesis. We also describe a general search strategy, called the *k-coherent search*, for fast BTF synthesis using surface textons. A BTF synthesized using our algorithm not only looks similar to the BTF sample in all viewing/lighthing conditions but also exhibits a consistent mesostructure when viewing and lighting directions change. Moreover, the synthesized BTF fits the target surface naturally and seamlessly. We demonstrate the effectiveness of our algorithm with sample BTFs from various sources, including those measured from real-world textures.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism–color, shading, shadowing, and texture; I.2.10 [Artificial Intelligence]: Vision and Scene Understanding–texture; I.3.3 [Computer Graphics]: Picture/Image Generation.

Keywords: Bidirectional texture function, 3D textons, reflectance and shading models, texture synthesis, texture mapping, surfaces

1 Introduction

Two main ingredients for visual realism are surface geometry and surface details. With recent advances in surface texture synthesis [20, 18, 22, 9], we can now decorate a real-world surface (e.g., reconstructed from laser range scans [4]) with a texture that fits the surface naturally and seamlessly. However, we are still a step away from reality because textures in traditional graphics represent only color or albedo variations on smooth surfaces. Real-world textures, on the other hand, arise from both spatially-variant surface reflectance and surface mesostructures, i.e., the small but visible local geometric details [11]. Mesostructures, which are responsible for fine-scale shadows, occlusions, and specularities [6, 12, 10, 13], are ignored by conventional textures. Fig. 1 compares a surface texture with a surface BTF.

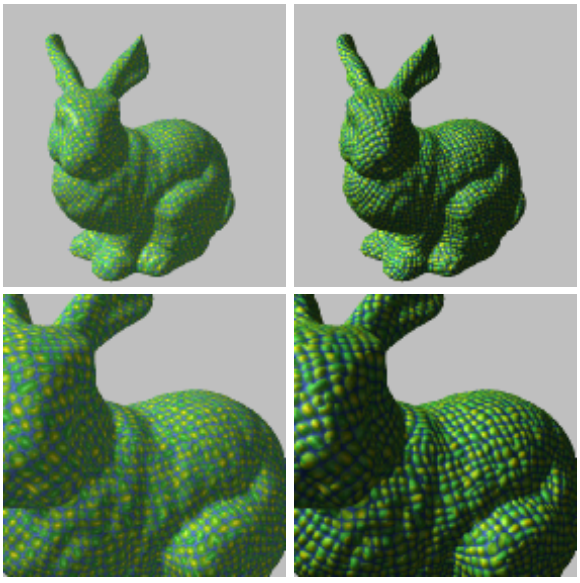

Figure 1: Left column exhibits images of a surface texture. Right column exhibits images of a surface BTF.

The BTF introduced by Dana et al. [6] is a representation of real-world textures that can model surface mesostructures and reflectance variations. The BTF is a 6D function whose variables are the 2D position and the viewing and lighting directions. In this paper, we present an algorithm for synthesizing the BTF on arbitrary polygonal surfaces. Given a BTF sample and a mesh, we synthesize a BTF on the mesh such that: (a) the surface BTF is perceptually similar to the given BTF sample in all viewing/lighting conditions, and (b) the surface BTF exhibits a consistent mesostructure when viewing and lighting directions change. The requirement of a consistent mesostructure is where surface BTF synthesis differs fundamentally from surface texture synthesis since conventional textures ignore mesostructures completely.

A BTF can be mapped onto surfaces using texture mapping techniques. However, BTF mapping on arbitrary surfaces can introduce inconsistent mesostructures. The usual technique for texture mapping arbitrary surfaces is to use a collection of overlapping patches [14, 16], and textures in the overlapping regions are blended to hide seams (e.g., see [16]). This technique works well for many textures [16], but for the BTF, blending can introduce inconsistent mesostructures, as Fig. 2 illustrates. Of course, BTF mapping on arbitrary surfaces also suffers from the usual problems of texture mapping, which include distortion, seams, and considerable user intervention needed for creating good-quality texture maps [20, 18, 22].

Liu et al. recognized the importance of mesostructures in developing their algorithm for synthesizing a continuous BTF from

*3F Beijing Sigma Center, No 49 Zhichun Road, Haidian District, Beijing 100080, P R China, email: bainguo@microsoft.com

†This research was done when Jingdan Zhang and Xi Wang were working as part-time interns at Microsoft Research Asia.

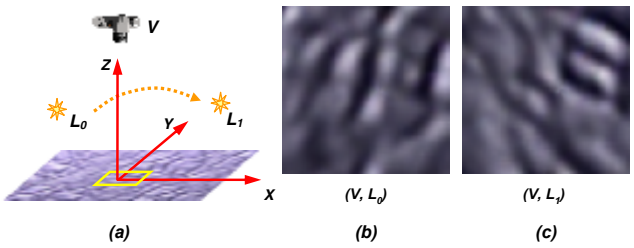

Figure 2: Inconsistent mesostructures caused by BTF blending. (a) The viewing/lighting setting for the images in (b) and (c). L_0 is the lighting direction for the image shown in (b) and L_1 is the lighting direction for the image shown in (c). Both images are for the square outlined by the yellow line in (a). The viewing direction for both images is the same.

sparse measurements [13]. Indeed, they explicitly recovered geometry details using a shape-from-shading method. The appearance of the recovered geometry is rendered and used for synthesis images under different viewing/lighting settings. Unfortunately, adapting [13] to surface BTF synthesis is not possible because it is time consuming to reconstruct/render the appearance from the recovered geometry for all lighting and viewing settings, especially for BTFs with complex geometry or radiance distributions. Moreover, current shape-from-shading techniques will have problems handling real-world textures with complex bump structures or without dominant diffuse components [13].

A possible way to achieve a consistent mesostructure on a surface is to directly apply surface texture synthesis techniques to surface BTF synthesis. The sample BTF may be regarded as a 2D texture map, in which the BTF value at a pixel is a 4D function of the viewing and lighting directions, and this 4D function can be discretized into a vector for texture synthesis. Unfortunately, this approach incurs a huge computational cost because of the large amount of data in a BTF sample. At the resolution of $12\times5\times12\times5$, the BTF value at a pixel is a 10800-dimensional vector, as opposed to the usual 3D RGB vectors [20, 18, 22]. Since texture synthesis time grows linearly with the vector dimension, a surface BTF can take days [20] or even months [18] to compute. Principal components analysis (PCA) can reduce the vector dimension somewhat but cannot alter the nature of the problem.

Leung and Malik introduced 3D textons based on the observation that, at the local scale, there are only a small number of perceptually distinguishable mesostructures and reflectance variations on the surface [12]. Their observation raises the hope of a compact data structure for extracting the essential information for surface BTF synthesis. However, 3D textons themselves are not compact because of their huge appearance vectors [12]. Indeed, the reconstructive BTF synthesis proposed in [12] is not feasible on surfaces because the basic operations [18] for surface BTF synthesis – texton resampling, distance computation, and BTF reconstruction – are prohibitively expensive with 3D texton [12]. To get a feel of the problem, consider the following: surface BTF reconstruction requires a 4D function to be assigned to each mesh vertex, which implies a 2.5 Gb storage for a mesh of 250k vertices [18].

In this paper, we show that surface BTF can be efficiently synthesized using the *surface textons*, which are derived from 3D textons [12]. Having no appearance vectors, surface textons constitute a compact data structure for extracting the essential information from the BTF sample to facilitate surface BTF synthesis. We present an algorithm based on surface textons for synthesizing a *surface texton map*, which is a texton-based compact representation of the synthesized surface BTF. This compact representation can be directly used for rendering; no BTF reconstruction like the one in [12] is necessary. We also describe a search strategy, called the *k-coherent search*, for fast surface BTF synthesis using surface textons. Existing general-purpose search strategies, such as those used in [20, 18, 9, 22], can also be adapted for surface textons, but the search speed is orders of magnitude slower.

We will demonstrate the effectiveness of our algorithm using surface BTFs synthesized from both real and synthetic BTF samples. With the increasing availability of BTF samples measured from real-world textures [6], surface BTFs provide a way to decorate real-world geometry with real-world textures. We also show that surface BTFs provide an efficient way for rendering surfaces with complex synthetic appearance models and geometry details.

The rest of the paper is organized as follows. After a brief review of related work in Section 2, we give an overview of our approach in Section 3. Section 4 discusses how to extract surface textons from the sample BTF. Section 5 provides details about surface BTF synthesis. Section 6 describes how to render surface BTFs synthesized by our algorithm. Results are reported in Section 7, followed by a conclusion and discussion about future work in Section 8.

2 Related Work

Several models exist for BTF analysis and material recognition, including the histogram model and correlation model for simple BTFs captured from random height fields with Lambertian reflectance [5], a color correlation model [10], a 3D-texton-based histogram model [12], and a model combining PCA and 2D textons [3].

Existing BTF synthesis methods [5, 12, 13] are designed for 2D rectangles [12] or rectangular surface patches [5, 13], not for arbitrary surfaces. Texture morphing [5] is a technique for BTF synthesis under the assumption that the surface is a simple height field with Lambertian reflectance. In [12], a BTF synthesis algorithm was suggested based on 3D textons. This algorithm first synthesizes a 2D map of texton labels using non-parametric sampling [8] and then reconstructs a BTF from the synthesized texton labels. A big problem with this reconstructive synthesis of a BTF is the high computational costs for synthesizing texton labels and reconstructing the BTF using the huge appearance vectors of textons (e.g., an appearance vector is 57600-dimensional if 400 sample images of the BTF are used as in [12]).

To capture and efficiently render the complex appearance of the real world surface, Malzbender et al. proposed polynomial texture maps for capturing the surface appearance under a fixed viewpoint but different lighting directions [15]. For surface light fields (e.g., [21]), the appearance of surfaces under fixed light sources but different view directions are captured and stored in a compact way for rendering.

Generating textures on arbitrary surfaces has been an active area of research (e.g., [14, 16, 20, 18, 22, 9]). One approach is to map texture patches onto the target surface [14, 16]. A good representative work following that approach is the lapped texture technique by Praun et al. [16]. They randomly paste texture patches onto the surface following orientation hints provided by the user. To hide the mismatched features across patch boundaries, textures in the overlapping regions are blended. This technique works well for many textures, but for highly structured textures and textures with strong low-frequency components, the seams along patch boundaries are still evident [16].

A number of algorithms have been proposed for directly synthesizing textures on arbitrary surfaces. Turk's algorithm [18], Wei and Levoy's algorithm [20], and the multi-resolution synthesis algorithm by Ying et al. [22] are general-purpose algorithms based on the search strategy proposed by Wei and Levoy [19]. The algorithm by Gorla et al. [9] is also a general-purpose algorithm, based on the search strategy proposed by Efros and Leung [8]. These algorithms

tend to be slow, but they can be accelerated by using either tree-structured vector quantization [19, 20] or a kd-tree [22]. Generally, these algorithms produce high-quality results. A special type of textures that cannot be handled well by general-purpose algorithms is the so-called "natural" textures [2]. Ashikhmin proposed a special-purpose algorithm for "natural" textures [2], and his algorithm has been adapted for "natural" surface textures [22]. However, [2, 22] do not generalize well to other types of textures.

3 Overview

Fig. 3 provides an overview of our system. The given sample BTF $T(x, y, \theta_i, \phi_i, \theta_r, \phi_r)$ is regarded as a texture map in which every pixel (x, y) has the value of a 4D function $T_{(x,y)}(\theta_i, \phi_i, \theta_r, \phi_r)$. Given T and a mesh M, we synthesize the surface BTF T' in two steps: texton analysis and surface BTF synthesis.

In the first step, we generate a 2D texton map t_{in} and build the surface texton space $\mathbf{S}$, which is represented by the dot-product matrix Γ. From the sample BTF T, we construct a 3D texton vocabulary $V = \{\mathbf{t}_1, ..., \mathbf{t}_{n_t}\}$ as in [12]. Based on V we can assign a texton label to each pixel of the sample BTF T and thus generate the 2D texton map t_{in}.

The surface texton space $\mathbf{S}$ is the inner-product space spanned by using the 3D textons $\{\mathbf{t}_1, ..., \mathbf{t}_{n_t}\}$ as basis vectors. Each element of $\mathbf{S}$ is called a surface texton. The surface texton space $\mathbf{S}$ is represented by the dot-product matrix Γ, an $n_t \times n_t$ matrix that stores the dot-product of every pair of 3D textons in V. The fact that there are only a small number of 3D textons [12] implies that the dot-product matrix Γ is compact, and so is $\mathbf{S}$. For example, a 64×64 BTF sample consisting of 3600 color images is about 59 Mb. Its representation with 400 3D textons extracted from 400 sample images is about 92Mb; the corresponding dot-product matrix is only 640 Kb. The construction of $\mathbf{S}$ is simple: all we need to do is calculate the dot-product matrix Γ and discard the appearance vectors.

In the surface BTF synthesis step, we use the surface texton map to compactly represent the surface BTF T'. The surface texton map t_{out} is a list of entries, one for each mesh vertex. The entry for vertex v, $t_{out}(v)$, consists of a texton label and a texture coordinate $p_v = (a_v, b_v)$, implicitly defining

$$T'_v(\theta_i, \phi_i, \theta_r, \phi_r) = T(a_v, b_v, \theta_i, \phi_i, \theta_r, \phi_r).$$

We treat the 2D texton map t_{in} as a texture sample and perform surface texture synthesis to generate the surface texton map t_{out}. We generate the surface texton map entries for mesh vertices incrementally, one vertex at a time. At each mesh vertex v, we simultaneously synthesize the texton label and generate a texture coordinate defining the BTF value at v.

The basic operations in surface texton map synthesis are texton resampling and the distance computation between surface textons. All these calculations can be carried out as operations in the surface texton space $\mathbf{S}$ and thus are fully determined by the pre-computed dot-product matrix Γ.

4 Texton Analysis

The texton analysis takes the following steps: (a) build a vocabulary of 3D textons from the sample BTF T, (b) assign texton labels to the pixels of T to get the 2D texton map t_{in}, and (c) construct the surface texton space $\mathbf{S}$ by calculating the dot-product matrix Γ and discarding the appearance vectors.

3D Texton Vocabulary: The construction of 3D textons is mostly based the original 3D texton paper by Leung and Malik [12]. As in

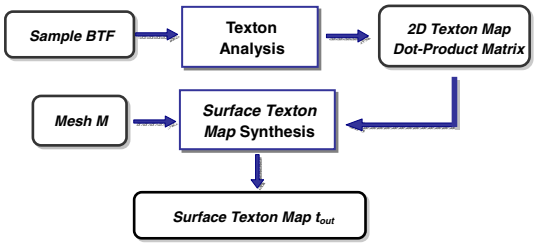

Figure 3: Data flow in our system.

[12], we construct 3D textons from a BTF using K-means clustering. To capture the appearance of mesostructures at different viewing/lighting conditions, we treat the BTF sample T_{in} as a stack of n images and filter each image with a filter bank of $n_b = 48$ Gaussian derivative filters. For each pixel of T_{in}, the filter responses of n_s selected images are concatenated into a $n_s n_b$-dimensional data vector. These data vectors are clustered using the K-means algorithm. The resulting K-means centers $\{\mathbf{t}_1, ..., \mathbf{t}_{n_t}\}$ are the 3D textons, and the associated $n_s n_b$-dimensional concatenated filter response vectors $\{\mathbf{v}_1, ..., \mathbf{v}_{n_t}\}$ are the appearance vectors. We also generate an extra texton by averaging the appearance vectors of all textons. This extra texton is used as the default texton in surface BTF synthesis. We use $n_t = 400$ for all examples in this paper.

Our 3D texton construction differs from [12] in choosing the n_s selected images. The reason for only selecting n_s images from T_{in} for clustering, where $n_s \ll n$, is to reduce computation by exploring the coherence of the same material in different viewing/lighting settings. In [12], the n_s images are randomly chosen. However, this is suboptimal because the radiance distribution is non-uniform in the viewing-lighting space. Cula and Dana [3] proposed a method for automatically selecting a subset representative images from a BTF sample. Their method works with a statistical representation of a BTF sample (the histogram of 2D textons). Because a histogram-based representation discards the spatial distribution of the BTF data, adopting [3] in BTF synthesis framework is difficult.

We choose the n_s representative images by K-means clustering in the viewing-lighting dimensions of the BTF sample T_{in}. Specifically, we filter image I_λ of T_{in} using our filter bank of 48 Gaussian derivative filters, producing a 48-dimensional filter-response vector for each pixel of I_λ. The filter-response vectors of pixels on a regularly-spaced subsampling grid in I_λ is concatenated into an image appearance vector representing I_λ. The image appearance vectors of all images in T_{in} are then K-means clustered. For each cluster, the image whose image appearance vector is nearest to the cluster center is selected as the representative image. Note that forming an image appearance vector by concatenating only filter-response vectors on a subsampling grid is the key to saving computation, and we are allowed to subsample because, as far as clustering is concerned, the filter-response vector of a pixel captures enough local structure around the pixel. We used $n_s = 400$ for all examples in this paper.

2D Texton Map: Once we have the texton vocabulary $\{\mathbf{t}_1,...,\mathbf{t}_{n_t}\}$, we can easily assign a texton label to each pixel of T_{in}. The texton label at pixel p is $t_{in}(p) = \arg\min_{j=1}^{n_t} \|\mathbf{v}(p) - \mathbf{v}_j\|^2$, where $\mathbf{v}(p)$ is the $n_s n_b$-dimensional concatenated filter response vector of pixel p, and $\mathbf{v}_j$ is the appearance vector of 3D texton $\mathbf{t}_j$. The resulting t_{in} is called a 2D texton map, or texton map for short.

Surface Textons: The 3D textons $\{\mathbf{t}_1, ..., \mathbf{t}_{n_t}\}$ can be regarded as abstract vectors and they span a vector space $\mathbf{S}$. Any vector $\mathbf{s}$ in $\mathbf{S}$ is of the form $\mathbf{s} = \sum_{i=1}^{n_t} a_i \mathbf{t}_i$, where $a_1, ...,$ and a_{n_t} are real numbers. We call $\mathbf{S}$ the surface texton space. The surface texton space is actually an inner-product space. The dot product of two basis vectors $\mathbf{t}_i$ and $\mathbf{t}_j$ is defined as $\mathbf{t}_i \cdot \mathbf{t}_j = \mathbf{v}_i \cdot \mathbf{v}_j$, where $\mathbf{v}_i$ and

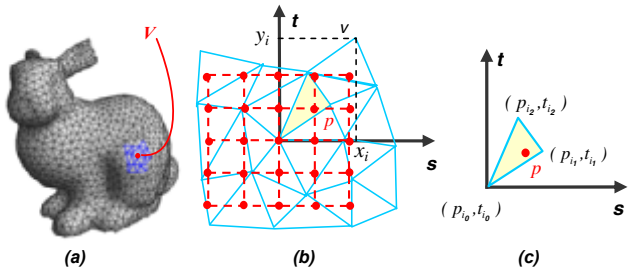

Figure 4: The surface patch around the red vertex v in (a) is flattened into the blue patch $P(v)$ in (b). The neighborhood template for resampling is shown in red in (b). The resampling of textons from vertices of the yellow patch triangle to a neighborhood pixel p is shown in (b) and (c).

$\mathbf{v}_j$ are the appearance vectors of $\mathbf{t}_i$ and $\mathbf{t}_j$ respectively. We precompute the dot product of every pair of basis vectors and store the results in an $n_t \times n_t$ matrix $\Gamma = (\gamma_{ij})$ such that $\gamma_{ij} = \mathbf{t}_i \cdot \mathbf{t}_j$. Once Γ is computed, we discard all appearance vectors.

An element of the surface texton space $\mathbf{S}$ is a surface texton. Note that $\{\mathbf{t}_1, ..., \mathbf{t}_{n_t}\}$, with their appearance vectors discarded, are also surface textons because they are the basis of $\mathbf{S}$. The resampling and distance computation for surface textons as required by surface BTF synthesis can be formulated as linear transformations and dot-products in the surface texton space $\mathbf{S}$. All these operations are abstract in that they do not refer to the appearance vectors. In particular, the dot product of any two vectors $\mathbf{s}$ and $\mathbf{s}'$ in $\mathbf{S}$ can be obtained easily from Γ, without referring to any appearance vector. Let $\mathbf{s} = \sum_{i=1}^{n_t} a_i \mathbf{t}_i$ and $\mathbf{s}' = \sum_{i=1}^{n_t} a'_i \mathbf{t}_i$. Then it is easy to verify that $\mathbf{s} \cdot \mathbf{s}' = \sum_{i,j=1}^{n_t} a_i a'_j \gamma_{ij}$.

5 Surface BTF Synthesis

5.1 BTF Synthesis with Surface Textons

Before the BTF synthesis starts, we have the 2D texton map t_{in} and the surface texton space $\mathbf{S}$ with the pre-computed dot-product matrix Γ. We define a local texture coordinate frame $(\vec{s}, \vec{t}, \vec{n})$ at each vertex of the target mesh M. The vector $\vec{n}$ is the surface normal at v, whereas $\vec{s}$ and $\vec{t}$ are the "right" and "up" directions determined by a vector field, which is either interpolated from a number of user-specified directions [18] or generated by relaxation [20].

Surface Texton Map Synthesis: A single-resolution version of the surface texton map synthesis proceeds as follows. We walk through the vertices of the target mesh M and compute an surface texton for every vertex. At each vertex v, the surface texton map entry $t_{out}(v)$ is obtained through the following steps. First, we construct a neighborhood $N(v)$ in the (s,t)-plane of v's local texture coordinate frame $(\vec{s}, \vec{t}, \vec{n})$. Then, we build a candidate set $C(v)$ consisting of the candidate pixels for v in the 2D texton map t_{in}. Next, we search the candidate set $C(v)$ to find a pixel $p_0 = (a_0, b_0)$ such that the distance between $N(v)$ and the neighborhood of p_0, $N(p_0)$, is the smallest. Finally, we set the texton label of the surface texton map entry $t_{out}(v)$ to be $t_{in}(p_0)$ and the texture coordinate of $t_{out}(v)$ to be (a_0, b_0). The pseudo-code of the surface texton map synthesis is as follows.

For each vertex v on surface
  construct neighborhood textons $N(v)$
  smallest_match = BIG;
  form the candidates set $C(v)$
  For each pixel $p = (a, b)$ in $C(v)$
    construct neighborhood textons $N(p)$
    new_match = $distance(N(v), N(p))$;
    If (*new_match* < *smallest_match*)
    smallest_match = *new_match*
    $t_0 = t_{in}(p), (a_0, b_0) = (a, b)$
  $t_{out}(v).texton\_label = t_0$
  $t_{out}(v).texture\_coordinate = (a_0, b_0)$

The surface texton map synthesis essentially generates the BTF value at vertex v by copying the BTF value at location p_0 in the sample BTF. Location p_0 is chosen according to the neighborhood similarity of $N(v)$ and $N(p_0)$ as measured by their surface textons. This is a valid similarity measure because the texton-based similarity of $N(v)$ and $N(p_0)$ implies their similarity as measured by their BTF values [12]. Of course a big advantage of the texton-based neighborhood similarity measure is that texton distances can be efficiently evaluated for surface textons.

Texton Resampling: Texton resampling is necessary for constructing neighborhood $N(v)$. We construct $N(v)$ in the (s,t)-plane of v's local texture coordinate frame $(\vec{s}, \vec{t}, \vec{n})$ as follows. First, a patch $P(v)$ is generated in the (s,t)-plane by flattening a set of triangles near v [16, 20]. Then, the pixels in the neighborhood $N(v)$ are resampled from the patch triangles using a neighborhood template [20], as is shown in Fig. 4. Finally, a surface texton $\mathbf{s}(p)$ is obtained at each neighborhood pixel p in $N(v)$ through the following interpolation:

$$\mathbf{s}(p) = w_0 \mathbf{t}_{i_0} + w_1 \mathbf{t}_{i_1} + w_2 \mathbf{t}_{i_2}, \quad (1)$$

where (w_0, w_1, w_2) is the barycentric coordinates of p in the patch triangle that contains p, and $\mathbf{t}_{i_0}$, $\mathbf{t}_{i_1}$, and $\mathbf{t}_{i_2}$ are textons at the vertices of that patch triangle. For implementation, $\mathbf{s}(p)$ can be efficiently represented by a 6-tuple $(w_0, w_1, w_2, \mathbf{t}_{i_0}, \mathbf{t}_{i_1}, \mathbf{t}_{i_2})$. The default texton is assigned to neighborhood pixels that are not contained by any patch triangle.

Distance Computation: We need to find a pixel $p_0 = (a_0, b_0)$ from the candidate set $C(v)$ such that the distance between the two neighborhoods, $N(v)$ and $N(p_0)$, is the smallest. For this purpose we need to compute, for each pixel p in $C(v)$, the distance between the neighborhoods $N(p)$ and $N(v)$. This distance can be written as

$$distance(N(v), N(p)) = \sum_{\lambda=1}^{n_v} ||\mathbf{s}(p_\lambda) - \mathbf{t}_{j_\lambda}||^2$$

where n_v is the number of pixels in $N(v)$ and each $\mathbf{s}(p_\lambda)$ is a surface texton. Each term $||\mathbf{s}(p_\lambda) - \mathbf{t}_{j_\lambda}||^2$ of the above distance can be written as the dot product of two surface textons,

$$(\mathbf{s}(p_\lambda) - \mathbf{t}_{j_\lambda}) \cdot (\mathbf{s}(p_\lambda) - \mathbf{t}_{j_\lambda}),$$

which can be easily evaluated using the pre-computed dot-product matrix Γ.

Multi-Resolution Synthesis: To improve synthesis quality, we have designed and implemented a two-pass multi-resolution version of our BTF synthesis algorithm in a fashion similar to [20, 18, 22]. In the pre-processing stage of the two-pass version, we build a texton pyramid and a mesh pyramid. For the texton pyramid, we first construct an image pyramid for each image of the BTF sample. Then a 2D texton map and a dot-product matrix is generated at each level. The number of textons at the next lower resolution l_{i+1} is about a quarter of that at the current resolution l_i. For the mesh pyramid, we use Turk's algorithm [17]. Starting from the highest resolution mesh, we generate the mesh in the next lower resolution level l_{i+1} by retiling the current level l_i mesh with about a quarter of the vertices. The vertices at each level of the mesh are randomly mapped to the pixels of the texton map at the same level, with the texton label and texture coordinate of a vertex coming from its corresponding pixel.

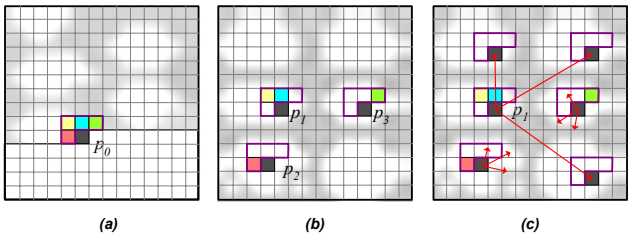

Figure 5: The k-coherence candidates of a pixel p_0 for $k = 4$. (a) The neighborhood $N(p_0)$ for pixel p_0 (colored black) in I_{out}. (b) The pixels of $C_1(p_0)$ in I_{in} are colored black. Each black pixel is the coherence candidate corresponding to a colored pixel in $N(p_0)$: p_3 is the coherence candidate corresponding to the green pixel, p_2 is the coherence candidate corresponding to the red pixel, and p_1 is the coherence candidate corresponding to both the yellow and blue pixels. (c) For each pixel in $C_1(p_0)$, its 3 nearest neighbors are added to $C_4(p_0)$. Here we only show the 3 nearest neighbors of p_1.

In the first pass, the surface texton map at the level l_i mesh is synthesized from the level l_{i+1} texton map. For a mesh vertex v_i at level l_i, we find a point v_{i+1} at the level l_{i+1} mesh by following the surface normal at v_i on the level l_i mesh. We compute the surface texton map entry at v_{i+1} using the level l_{i+1} texton map. The texture coordinate of v_i is derived from that of v_{i+1}. The texton label at v_i is fetched from the level l_i texton map using v_i's texture coordinate.

In the second pass, when synthesizing the surface texton map entry at vertex v_i in the level l_i mesh, we use the neighborhood of v_i as well as that of v_{i+1} at level l_{i+1}, where v_{i+1} is found as in the first pass. For vertex v_i, we form the candidate set $C(v_i)$ using v_i's neighborhood at level l_i only. The two-level neighborhoods and the corresponding dot-product matrices are used for neighborhood distance computation when searching for the best candidate from $C(v)$.

5.2 Fast Search for Surface Textons

So far we have not described the search strategy, i.e., the strategy to form the candidate set $C(v)$ for a mesh vertex v. Most existing search strategies [20, 18, 22, 9] can be adapted for constructing $C(v)$. We have experimented with the full search, in which $C(v)$ consists of all pixels of the 2D texton map t_{in}. This is a general search strategy used by [20, 18, 9] and the multiresolution synthesis algorithm in [22]. Unfortunately, the full search is painfully slow with surface textons for two reasons. First, the full search is itself slow because the candidate set is as big as it gets. Second, existing acceleration techniques including vector quantization [19] and the kd-tree [1] do not work well with surface textons because surface textons are not the usual intensity values. The kd-tree, for example, requires sorting data vectors by one of their components [1]. Such sorting is not possible when the data vectors are surface textons. For fast searching with surface textons, we have developed a general search strategy called the k-coherence search.

Candidate Set for 2D Textures: For simplicity we explain the k-coherence search in the context of synthesizing a 2D texture I_{out}. Suppose we are to synthesize a pixel p_0 of I_{out} based on the already synthesized pixels in a neighborhood $N(p_0)$ of p_0, as is illustrated in Fig. 5. Every synthesized pixel p_s in $N(p_0)$ corresponds to a pixel p_1 in the input sample texture I_{in}. We call p_1 a coherence candidate for p_0 because it is a good candidate according to the coherence of I_{out}: a pixel that is appropriately "forward-shifted" with respected to a pixel already used for synthesis is well-suited to fill in p_0 [2]. The coherence candidates are collected in $C_1(p_0)$, the coherence candidate set.

The k-coherence search constructs the candidate set $C(p_0)$ as the k-coherence candidate set $C_k(p_0)$, which is formed by adding, for each pixel p_1 of $C_1(p_0)$, a set of pixels $\{p_2, ..., p_k\}$ of I_{in} such that the newly-added pixels are closer to p_1 than any other pixels in I_{in} by the neighborhood distance. The idea of k-coherence search is to speed up the search by guiding it to pixels of I_{in} that are close to the coherence candidates according to the neighborhood distance. This guidance is valid because by the Markov property [8, 19], whether a pixel is an eligible candidate is completely determined by pixels in its surrounding neighborhood. If the coherence candidates are suitable to fill p_0, then pixels close to the coherence candidates by the neighborhood distance are also good candidates for p_0.

The k-coherence search is fast because the k-coherence candidates set is much smaller (usually $k \leq 11$) than that of the full search and it can be constructed very quickly with the pre-computed list of k nearest neighbors for each pixel of I_{in}. For a small I_{in} ($\leq 64 \times 64$), the k nearest neighbors of every pixel of I_{in} can be pre-computed fairly quickly by an exhaustive search in I_{in}. For a large I_{in} ($> 64 \times 64$), a two-level pyramid is built to speed up the pre-processing of lists of k nearest neighbors for all pixels in I_{in}. Specifically, to compute the k nearest neighbors of a pixel $p(a, b)$, we first compute m initial candidates for $p(a/2, b/2)$ in the low-resolution version of I_{in}, where $m = 100$ in our implementation. For each initial candidate in the low-resolution version of I_{in}, its four corresponding pixels in I_{in} are added to the set of initial candidates in I_{in}. After all $4 * m$ initial candidates are so generated, the k nearest neighbors of pixel p are found from these initial candidates.

An important advantage of the k-coherent search is that its pyramid-based acceleration also works for surface textons. For the k-coherent search, the low-pass filtering needed for pyramid-based acceleration only takes place on the 2D texton map t_{in}. The texton pyramid constructed for multi-resolution synthesis can also be used for building the list of the k nearest neighbors. As a result, we do not need to low-pass filter the surface textons during the surface texton map synthesis. Low-pass filtering surface textons is a hard operation to define because surface textons have no appearance vectors.

Fig. 6 shows the basic behaviors of k-coherence search. When $k = 11$ the results of the k-coherence search are practically the same as that by the full search [19]. As k decreases, the results look less and less like that of the full search. When $k = 1$, the results become the same as those generated by Ashikhmin's algorithm [2].

Candidate Set on Surfaces: We now consider the construction of the k-coherence candidate set $C_k(v)$ for a mesh vertex v. Let $\{v_1, ..., v_m\}$ be the set of all vertices in the flattened patch $P(v)$ whose surface textons have been synthesized. Vertex v_i has a texture coordinate (s_i, t_i) and an offset (x_i, y_i) from v in the patch $P(v)$. As shown in Fig. 4 (b), we forward-shift (s_i, t_i) by the offset (x_i, y_i) in the 2D texton map t_{in}, getting to location $(s_i', t_i') = (s_i - x_i, t_i - y_i)$ in t_{in}. Then we fetch the list L_i of k nearest neighbors at the pixel closest to (s_i', t_i'). The candidate set $C_k(v)$ consists of all k nearest neighbors in all the lists L_1 through L_m.

In multiresolution synthesis, a list of k nearest neighbors is built for each pixel of the texton map at every level. In the second pass of a two-pass synthesis, we also use a two-level neighborhood when building the list of k nearest neighbors for every pixel so that the neighborhoods on the side of the texton pyramid are consistent with the two-level neighborhoods on the side of the mesh pyramid.

Discussion: The k-coherence search was inspired by Ashikhmin's work [2]. However, our goal is different from Ashikhmin's. We want to derive a general-purpose search strategy for fast search with surface textons; his goal was to develop a special-purpose algorithm for handling "natural" textures, i.e., textures consisting of arrangements of small objects of familiar but irregular shapes [2].

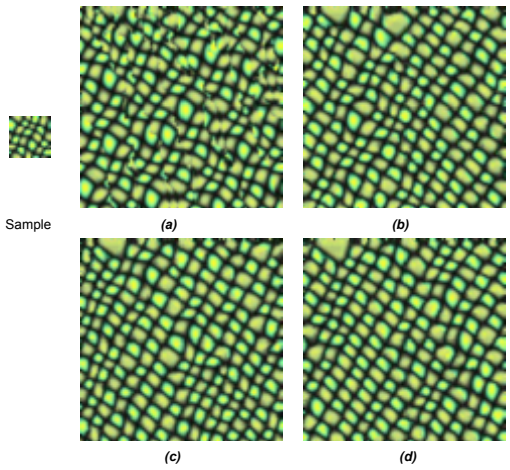

Figure 6: 2D texture synthesis results using the k-coherence search and the full search. (a) k-coherence search with $k = 1$. (b) k-coherence search with $k = 5$. (c) k-coherence search with $k = 11$. (d) The full search [19].

6 Surface BTF Rendering

From the surface texton map t_{out} and the sample BTF T, we can efficiently render the BTF on the target mesh M as follows. First, we compute the viewing and lighting directions for each mesh vertex v in its local texture coordinate frame from the given light source location and the viewpoint. Vertices occluded from either the light sources or the viewpoint are ignored. Then, a set of nearby images are found from the BTF sample T. Using v's texture coordinate, we can look up colors from this set of images and blend them to get the color of v. With all vertex colors obtained, the mesh can be sent to the graphics pipeline for display. This procedure repeats for every novel lighting/viewing configuration.

Finding the nearest images from the sample BTF T is simple because the images in T are evenly distributed in the viewing and lighting space. We first find 4 nearest sample viewing directions and 4 nearest sample lighting directions separately. The angle between two lighting/viewing directions is used as the distance measure. Then, the 4×4 nearest images are simply those corresponding to all combinations of the viewing/lighting directions found in the previous step. Debevec et al. [7] proposed a general technique for finding the nearest images for a given viewing/lighting setting.

7 Results

We have implemented our surface BTF synthesis algorithm on a PC. The system is easy to use. The texton analysis stage involves no user intervention. The BTF synthesis stage is as automatic as surface texture synthesis (e.g., [18, 20]). The system only requires the user to determine whether the BTF sample resemble "natural" textures and if so, set $k = 1$ for the k-coherence search. Anisotropic BTFs are handled the same way as anisotropic textures in surface texture synthesis [18, 20]. Like [18, 20], an optional but often helpful user intervention for an anisotropic BTF is to specify a vector field to guide the orientation of the BTF on the target surface.

In the following we report synthesis results for both sample BTFs of real-world textures and synthetic BTF samples. The real-world samples were taken from the CUReT database [6]. All surface BTFs shown in this paper were synthesized using three- or four-level pyramids of the meshes and the sample BTFs. All examples use k-coherence search with $k = 11$ unless otherwise mentioned.

Model	Sample Size	Vertex Number	Time (minutes)
Dinosaur	96×96	250k	21 (k=1)
Horse	128×128	250k	22 (k=1)
Cat	64×64	300k	141
Bunny	128×128	300k	186

Table 1: Timings for synthesizing the surface BTFs shown in Fig. 7.

Sample Size	Full Search	k-Coherence Search
64×64	747 min.	70 min.
96×96	3000 min.	123 min.
128×128	8066 min.	157 min.

Table 2: Speed comparison for BTF synthesis using the full search (e.g. [18]) and the k-coherence search.

Fig. 7 exhibits several surface BTFs. Timings for surface BTF synthesis are summarized in Table 1. Timings are in minutes measured on a 700MHz Pentium III. The time complexity of our algorithm only depends on size of the neighborhood and k for the k-coherence search. During the BTF synthesis, our algorithm uses about the same amount of memory as surface texture synthesis algorithms such as [20, 18]. The only extra memory we need is that for the dot-product matrix, which is less than 1 Mb in our system. Timings are for BTF synthesis only. The texton analysis time for a BTF sample of size 64×64 takes about 45 minutes on the same machine. For the BTF sample in Fig. 7 (a) and (b), we set $k = 1$ because the samples resemble "natural" textures [2].

Like surface textures, the synthesized BTFs fit the surface geometry naturally and seamlessly. More importantly, the surface BTFs capture the fine-scale shadows, occlusions, and specularities caused by surface mesostructures. Comparison of the synthesized surface BTFs with the sample BTFs demonstrates their similarity. In the companion video, we show that this similarity remains in all viewing and lighting conditions. Moreover, the synthesized mesostructures are consistent as viewing and lighting directions change.

Table 2 compares the BTF synthesis speed for the full search (e.g. [18]) and the k-coherence search using the dinosaur model with 250k vertices. For a small BTF sample (64×64), BTF synthesis based on the k-coherence search is about 10 times faster than that based on the full search. The speed gain increases quickly as the BTF sample gets larger. In Fig. 9, we compare the qualities of the surface BTFs synthesized with these two search strategies by rendering the BTFs with identical viewing and lighting. Our experiments demonstrate that the k-coherence search allows us to synthesize surface BTFs orders of magnitude faster while getting comparable quality.

Surface BTFs provide an efficient way for rendering surfaces with complex appearance models that are created synthetically. Fig. 8 shows an example, in which a small BTF sample is rendered by ray tracing and then synthesized onto a surface. The rendering of the surface BTF captures the shadow and occlusion caused by the height field of the synthetic appearance model. Conventional textures and bump maps cannot capture these effects. Although displacement maps can capture these effects, rendering displacement maps is very expensive. In addition, some appearance models have complex local geometry details and BRDF variations that cannot be rendered by displacement maps. Surface BTFs will not have problems handling these appearance models. With our unoptimized implementation, the surface BTF shown in Fig. 8 (300k vertices) can be rendered at a speed of more than one frame per second.

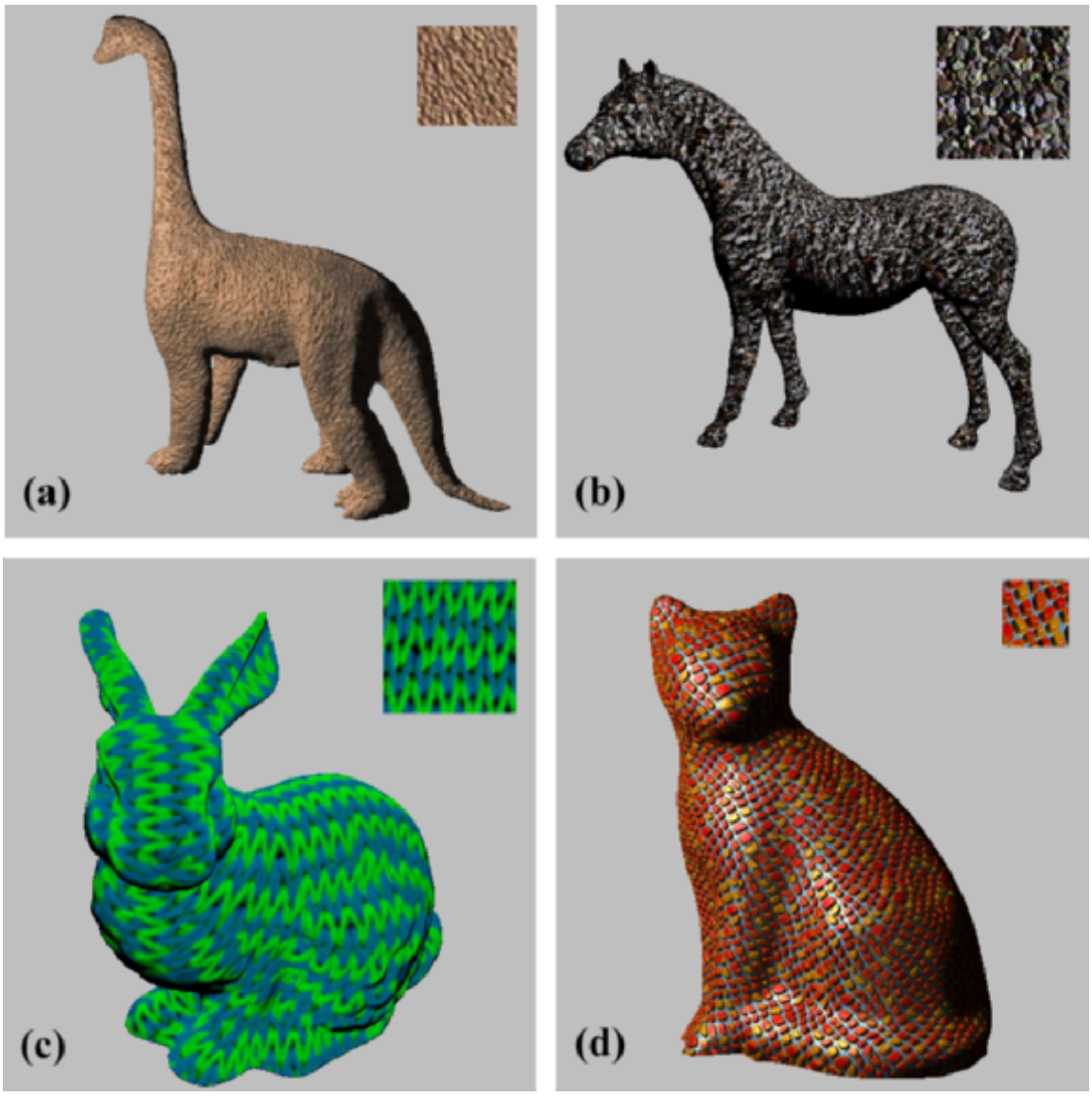

Figure 7: Top row: Surface BTFs synthesized using real BTF samples from the CUReT database [6]. Bottom row: Surface BTFs synthesized from synthetic BTF samples.

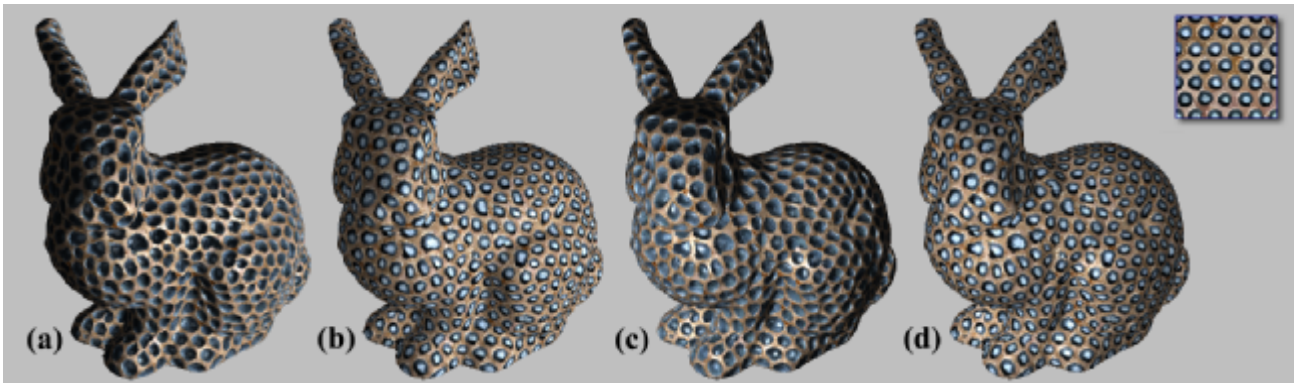

Figure 8: (a) Surface BTF. (b) Surface texture. (c) Surface BTF. (d) Surface texture.

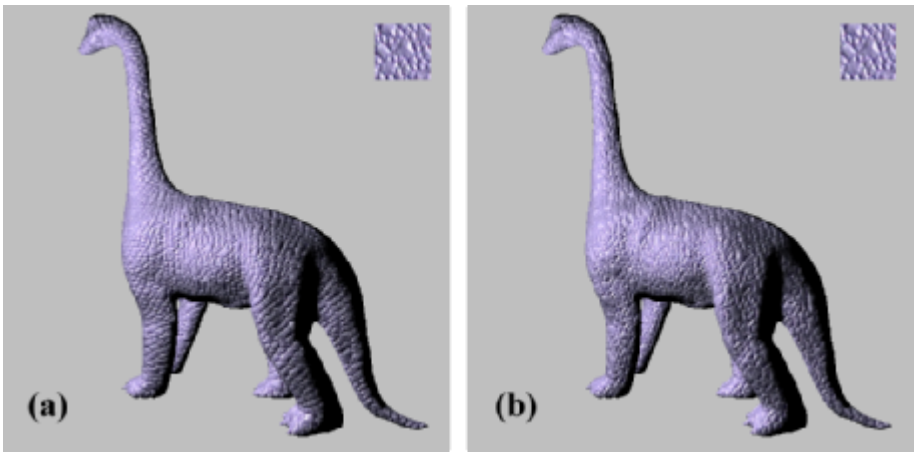

Figure 9: Quality comparison for BTF synthesis using the k-coherence search (left) and the full search (right).

8 Conclusion

We have presented an algorithm for synthesizing BTFs on arbitrary manifold surfaces using surface textons. A BTF synthesized using our algorithm not only looks similar to the sample BTF in all viewing and lighting conditions but also exhibits a consistent mesostructure when the viewing and lighting directions change. Because the BTF can describe real-world textures, our algorithm enables the user to decorate real-world geometry with real-world textures.

A limitation of surface BTF synthesis using surface textons is that the synthesis algorithm does not work well for materials that cannot be described by 3D textons. So far one such material has been reported [12], i.e., the "Aluminum Foil" dataset in the CUReT database [6]. It will be desirable to develop more sophisticated filtering/clustering techniques for this sort of materials. We are also interested in efficient rendering methods for a surface BTF represented by a sample BTF and a surface texton map. The basic rendering operations of a surface BTF so represented are simple and should be amenable to hardware acceleration. Finally, an intriguing possibility is to use synthesis methods to produce appearance not captured by the BTF, e.g., subsurface scattering.

Acknowledgments: We would like to thank Yanyun Chen and Xinguo Liu for useful discussions. Many thanks to Yanyun Chen for his help in generating synthetic BTF data, to Yin Li, Gang Chen, and Steve Lin for their help in video production, to Steve Lin for proofreading this paper, and to anonymous reviewers for their constructive critique. Xiang Cao implemented the first version of 3D textons.

References

[1] Sunil Arya, David Mount, Nathan Netanyahu, Ruth Silverman, and Angela Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.

[2] Michael Ashikhmin. Synthesizing natural textures. *2001 ACM Symposium on Interactive 3D Graphics*, pages 217–226, March 2001.

[3] Oana G. Cula and Kristin J. Dana. Compact representation of bidirectional texture functions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, December 2001.

[4] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 303–312, New Orleans, Louisiana, August 1996.

[5] Kristin J. Dana and Shree Nayar. 3d textured surface modeling. In *Proceedings of IEEE Workshop on the Integration of Appearance and Geometric Methods in Object Recognition*, pages 46–56, June 1999.

[6] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics*, 18(1):1–34, January 1999.

[7] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. *Eurographics Rendering Workshop 1998*, pages 105–116, June 1998.

[8] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of International Conference on Computer Vision*, September 1999.

[9] Gabriele Gorla, Victoria Interrante, and Guillermo Sapiro. Growing fitted textures. *SIGGRAPH 2001 Sketches and Applications*, page 191, August 2001.

[10] Pei hsiu Suen and Glenn Healey. The analysis and recognition of real-world textures in 3d. *IEEE Transactions on Patten Analysis and Machine Intelligence*, 22(5):491–503, May 2000.

[11] Jan J. Koenderink and Andrea J. Van Doorn. Illuminance texture due to surface mesostructure. *Journal of the Optical Society of America*, 13(3):452–463, 1996.

[12] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using 3d textons. *International Journal of Computer Vision*, 43(1):29–44, June 2001.

[13] Xinguo Liu, Yizhou Yu, and Heung-Yeung Shum. Synthesizing bidirectional texture functions for real-world surfaces. *Proceedings of SIGGRAPH 2001*, pages 97–106, August 2001.

[14] Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. *Proceedings of SIGGRAPH 93*, pages 27–34, August 1993.

[15] Tom Malzbender, Dan Gelb, and Hans Wolters. Polynomial texture maps. *Proceedings of SIGGRAPH 2001*, pages 519–528, August 2001.

[16] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. *Proceedings of SIGGRAPH 2000*, pages 465–470, July 2000.

[17] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):55–64, July 1992.

[18] Greg Turk. Texture synthesis on surfaces. *Proceedings of SIGGRAPH 2001*, pages 347–354, August 2001.

[19] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. *Proceedings of SIGGRAPH 2000*, pages 479–488, July 2000.

[20] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. *Proceedings of SIGGRAPH 2001*, pages 355–360, August 2001.

[21] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3d photography. In *Proceedings of SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 287–296, July 2000.

[22] Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. Texture and shape synthesis on surfaces. *Proceedings of 12th Eurographics Workshop on Rendering*, pages 301–312, June 2001.