Automatic Synthesis of Isotropic Textures on Surfaces from Sample Images

Joel Castellanos and Lance R. Williams

Abstract—A fully automatic method of synthesizing isotropic textures on subdivision surfaces from sample images is presented. Both Gaussian and Laplacian pyramid representations of the sample texture are constructed. Texture synthesis proceeds coarse-to-fine, by incrementally inverting the Laplacian pyramid to produce an initial guess and refining this guess using non-parametric sampling. The sampling procedure uses a nearest neighbor search while preserving first-order statistics. The resulting texture is generated directly on the subdivision surface. Within the domain of isotropic textures, the proposed method offers improvements in faithful reproduction of a sample's appearance over a wide range of scales. The method can also be used to produce isotropic variants of anisotropic textures. Finally, while the sampling procedure we describe is not amenable to standard methods for nearest neighbor search in high dimensional spaces, an acceleration method is proposed that uses an eigenvector transform and a set of dynamic Kd-trees.

Index Terms— B.1.2.a Automatic synthesis, I.4.6.c Markov random fields, I.4.7.f Texture, I.4.10.d Statistical.

◆

1 INTRODUCTION

1.1 Texture

Texture can be thought of as a visual pattern that is spatially repeated either deterministically or stochastically. The pattern may be completely deterministic, such as a chessboard, completely random, such as well-mixed salt and pepper, or exhibit intermediate degrees of randomness, such as tree bark, clouds, or marble. Textures can also be classified on a continuum from isotropic (having no particular orientation) to strongly anisotropic. Polished granite, for example, is isotropic, while a texture comprised of horizontal layers of sandstone is anisotropic. With the trivial exception of uniform colors, deterministic textures are anisotropic. A texture may exhibit different properties at different scales – see Fig. 1.

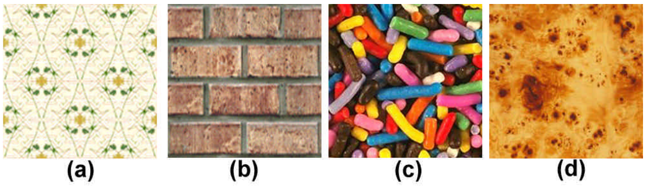


Fig. 1. Example Textures: (a) Floral print fabric that is deterministic and hence anisotropic. (b) Brick wall having a large-scale structure that is anisotropic; however, at small scales (within a single brick or mortar line), the structure is isotropic. (c) Texture formed by the random arrangement of identical elements exhibit aspects of both determinism and randomness, yet are isotropic at all scales. (d) Some textures can only be characterized statistically. Burl wood is an example of such a texture. It is isotropic at all scales.

1.2 Texture Synthesis

Texture synthesis is the creation of a new texture that is different from a sample, yet can be said to have the same visual appearance. Texture synthesis has been an active research topic in computer vision and graphics. One of he most straightforward applications is to synthesize a texture on the surface of an object in a computer generated scene in order to increase its degree of visual realism. Other image-based applications include inpainting, foreground removal, [6], lossy image and video compression, and animation of a static texture [1]. Fig. 2 shows an example of texture synthesis using the method proposed in this paper.

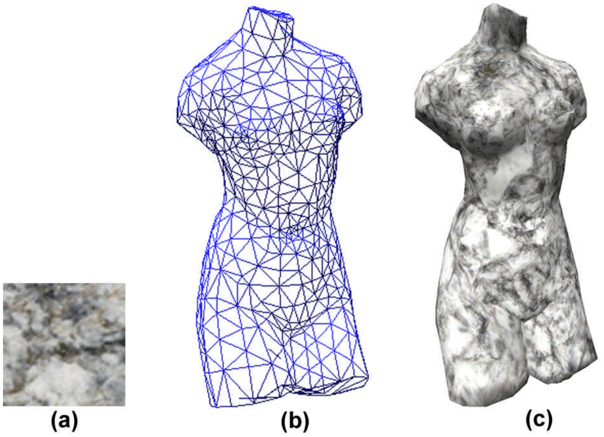


Fig. 2. Texture Synthesis on a Surface: (a) Input sample image. (b) Input triangulated surface of Torso of Venus de Milo with 1,418-faces. (c) View of synthetic texture synthesized using the method described in this paper.

Two major approaches to texture synthesis are procedural and sample-based. Procedural synthesis generates textures from mathematical equations that are particular to each texture. While procedural synthesis is currently very popular, a disadvantage is that the parameters yielding in a particular texture can be difficult to discover. Designers can spend weeks tweaking parameters, and the results, while possibly quite interesting and beautiful, are generally unpredictable. By contrast, sample-based synthesis applies a single procedure to a wide range of given texture samples.

- *J. Castellanos is a graduate student in the Dept. of Computer Science, University of New Mexico, Albuquerque, NM. E-mail: joel@cs.unm.edu.*
- *L.R. Williams is an Associate Professor in the Dept. of Computer Science, Univ. of New Mexico, Albuquerque, NM. E-mail: williams@cs.unm.edu.*

1.3 First and Second Order Statistics

Throughout the literature, it is common to refer to the first-order and second-order statistics of a texture. With few exceptions (mostly quite contrived), a pair of textures with the same first and second order statistics appear equivalent to the human observer. First-order statistics are the red, blue, and green pixel value histograms. These depend only on individual pixel values and not on the interaction or co-occurrence of neighboring pixel values. Second-order statistics are defined as the probability of observing a pair of color values occurring at the endpoints of a dipole of random length placed in the image at a random location and orientation.

1.4 Quilting

Quilting methods are sample-based texture synthesis methods that stitch together representative patches of a sample texture. Alpha blending and more sophisticated methods including dynamic programming are used to eliminate discontinuities at patch boundaries. Methods often include heuristics for automatically selecting “good” patches from a sample. For highly structured textures, the patch boundaries are generally chosen to avoid cutting across important features. For more stochastic textures, irregularly shaped patches generally work best. Quilting methods such as those described by Praun [9] and Efros and Freeman [5] can be very fast and produce excellent results for a wide range of textures on parameterized surfaces. There are, however, disadvantages inherent to any quilting process. First, quilting methods cannot reproduce structure at scales significantly larger than the size of the quilting elements, nor can they reproduce stochastic properties that occur on a scale smaller than the quilting elements. For example, in a texture of pebbles, there may be one odd shaped pebble with a distinctive marking. Quilting algorithms are likely to produce many copies of this odd shaped pebble with the same distinctive marking.

When applying quilting methods to surfaces, there is a third inherent disadvantage: Since quilting methods have fewer degrees of freedom to exercise than pixel synthesis methods, frustration due to misalignment of quilting patches on a surface can only be relaxed by repositioning and reorientation the patches. By contrast, pixel based texture synthesis methods can relax this frustration more uniformly. This effect is most noticeable when quilting anisotropic textures onto a surface lacking an explicit two-dimensional parameterization. This results in visible discontinuities. Therefore, quilting a surface with anisotropic elements requires that a two-dimensional parametrization be imposed on the surface, and when parameterizing most surfaces, the creation of singularities is unavoidable.

1.5 Neighborhood Search Methods

Neighborhood search methods of texture synthesis are based on a Markov random field texture model. This model makes the assumption that the probability distribution of brightness values for a pixel are conditioned solely on the brightness values of pixels in its neighborhood and are independent of those in the rest of the image. The Markov random field is used to model a texture's second order statistics.

Neighborhood search methods synthesize an output texture pixel by pixel given a sample input image. To determine the value of a particular output pixel, its spatial neighborhood is compared against all possible neighborhoods from the input image. Its value is then replaced with the value of the input pixel with the most similar neighborhood. Texture synthesis methods based on neighborhood search have used update orders that are scanline, spiral, surface sweeping, and random. In order to capture fine detail, a single pixel must be used as the unit of synthesis, yet the pixel's neighborhood must be large enough to capture a texture's structure at the largest scale represented in the sample. Since search time increases rapidly with neighborhood size, neighborhood search methods can be very slow. Multi-resolution formulations of neighborhood search methods synthesize texture using relatively small neighborhoods with successively finer resolution. Large-scale structure is captured by low-resolution neighborhoods and fine-scale structure is captured by high-resolution neighborhoods. Multi-resolution neighborhood search methods, therefore, have the advantage of being much faster than otherwise equivalent single resolution methods.

2 PREVIOUS WORK

2.1 Efros and Leung

The sample-based, texture synthesis method of Efros and Leung [6], can be viewed as a non-parametric sampling procedure for an implicit Markov random field. Their algorithm "grows" texture, pixel by pixel, outwards from an initial seed. All previously synthesized pixels in a square window around a given pixel (weighted to emphasize proximity) are used as the context. The size of the window is a free parameter that specifies the degree of randomness of a given texture. To synthesize a pixel, the algorithm first finds all neighborhoods in the sample image that are "similar" to the pixel's neighborhood. It then randomly chooses one of these neighborhoods and takes its center value to be the new center value of the output neighborhood.

2.2 Wei and Levoy

Our work builds most directly upon the multi-resolution, neighborhood-search-based method of Wei and Levoy [13]. For non-parameterized surfaces, Wei and Levoy's method takes two inputs: 1) A sample image; and 2) A triangulation of the surface on which the texture is to be synthesized. They build a partial a Gaussian pyramid from the sample image with K levels (usually with K=4). The triangulation representing the geometry of the surface is then recursively subdivided K times with subdivision k corresponding to level k of the Gaussian pyramid. Texture synthesis proceeds coarse-to-fine in three passes for each level in the Gaussian pyramid being synthesized. In the first pass of level k, the level is initialized with randomly chosen RGB values from the corresponding level of the sample's Gaussian pyramid (this matches the first-order statistics of level k). The second pass propagates information from coarse scales to fine scales using a neighborhood search process based on a neighborhood spanning two levels of the pyramid: the current level and next coarser level (levels k and k-1). The third pass on level k, further refines each face's RGB values using a neighborhood search process based on a neighborhood consisting solely of pixels within level k.

3 OUR APPROACH

3.1 Overview

The method we describe is restricted to isotropic textures because only isotropic textures can be automatically generated on an arbitrary surface in the absence of a parametrization.

Our method has been inspired by other multi-resolution, texture synthesis methods that non-parametrically sample an implicit Markov random field. Although our primary inspiration was Wei-Levoy[13], we have also been influenced by Heeger-Bergen[7], Efros-Leung[6], Debonet[4], and Bar-Joseph[1]. Like Wei-Levoy[13] we synthesize texture in a coarse-to-fine manner by searching for the face in the analysis Gaussian pyramid with the most similar neighborhood. However, our method differs from Wei-Levoy[13] in three significant ways. First, our method propagates information from coarse-to-fine in a more principled, and more effective manner, namely, by incrementally inverting a Laplacian pyramid transform. A second difference is that, the greedy method of non-parametric sampling described by Wei-Levoy[13] has been replaced with a sampling procedure of our own design that does a better job of preserving the first-order statistics of the sample. Finally, as a pre-processing step, we resample the input texture using a triangular grid so that its geometry more closely resembles that of the subdivision surface on which the texture will be synthesized. Representing both the analysis and synthesis pyramids using the same geometry greatly simplifies the neighborhood matching procedure that is the heart of the texture synthesis process.

3.2 Coarse-to-fine Information Flow

The proposed method computes both Gaussian and Laplacian pyramid representations of the sample texture. Having both representations of the sample allows the synthesis process to propagate information from coarse scales to fine scales by incremental inversion of the Laplacian pyramid transform. The procedure we use to implicitly sample the Markov random field at level k of the analysis Gaussian pyramid returns not only the Gaussian RGB values, as in Wei & Levoy [13], but also the RGB values of the four Laplacian children at level k+1 (the next finer resolution level). Therefore, we simultaneously synthesize Gaussian level k, (symbolized G_k), and Laplacian level k+1 (symbolized L_{k+1}). Once G_k and L_{k+1} are complete, we compute a first approximation of G_{k+1} by upsampling G_k, convolving it with a Gaussian kernel, and adding this to L_{k+1}. This incremental inversion of the Laplacian pyramid produces a very smooth coarse-to-fine information flow where large-scale structures, which form in coarser resolution levels, propagate to finer resolution levels.

Only the coarsest level of the synthesis Gaussian pyramid, G_0, is initialized with the RGB values of randomly chosen faces of the coarsest level of the analysis Gaussian pyramid. After a given level is initialized (by random samples in the case of G_0, or by incremental inversion of the Laplacian pyramid for all other levels), each face is visited in random order and its value is asynchronously updated using a non-parametric sampling procedure.

Fig. 3 illustrates the information flow of the process. Notice that low frequency structures that are present in the final texture (i.e., the green/yellow region and the blue region in Step 14) form in the first step of the sampling (G_0 in Step 8). By contrast, the results of Wei-Levoy[13] (who initialize each level with random values from the sample's pyramid) exhibit large shifts in spatial structure as synthesis progresses from coarse to fine.

3.3 Laplacian Pyramid Transform

The Laplacian pyramid transform is a linear transform that represents a gray scale image by decomposing it into a set of bandpass images, and a single residual lowpass image [3]. The original image can be perfectly reconstructed from the set of bandpass images, together with the residual lowpass image. Color images can be represented by transforming each color channel independently. The Laplacian pyramid transform is a historical predecessor of and closely related to the fast wavelet transform.

Each iteration of the process proceeds as follows: 1) The image is lowpass filtered by convolution with a Gaussian kernel; 2) The result is then subtracted from the original image. This difference image is called the bandpass image and retains the high-frequency detail. The bandpass image can be thought of as a second derivative of the lowpass image; 3) The lowpass image that resulted from the first step is then downsampled by a factor of two (every other pixel in both dimensions is discarded).

The three steps described above are repeated recursively, using the result of the third step at one level as input to the first step at the next level. In the case of a square input image of size 2^K, then the process can be repeated K times giving a residual lowpass image consisting of a single pixel. The lowpass images are called the Gaussian pyramid levels, G_k. Let G_{K-1} be the original image and G_0 be the residual lowpass image. The bandpass images are called the Laplacian pyramid levels, L_k.

The transform is inverted in the reverse order: 1) Each Gaussian pyramid level, G_k, beginning with the lowest resolution, G_0, is upsampled (zeros are inserted between sample points in both dimensions); 2) The upsampled Gaussian level is interpolated by convolution with the same Gaussian kernel used in the encoding process; 3) The result is added to the corresponding Laplacian pyramid level, L_{k+1}. As with encoding, the process is repeated for each level of the pyramid. This decoding process reconstructs the original image without error.

For notational convenience we define two operations: *reduce* and *project*. The reduce operation is Gaussian convolution followed by downsampling. The project operation is upsampling followed by Gaussian convolution:

$$G_{k-1} = reduce(G_k) \tag{1}$$

$$L_{k-1} = G_k - project(G_k) \tag{2}$$

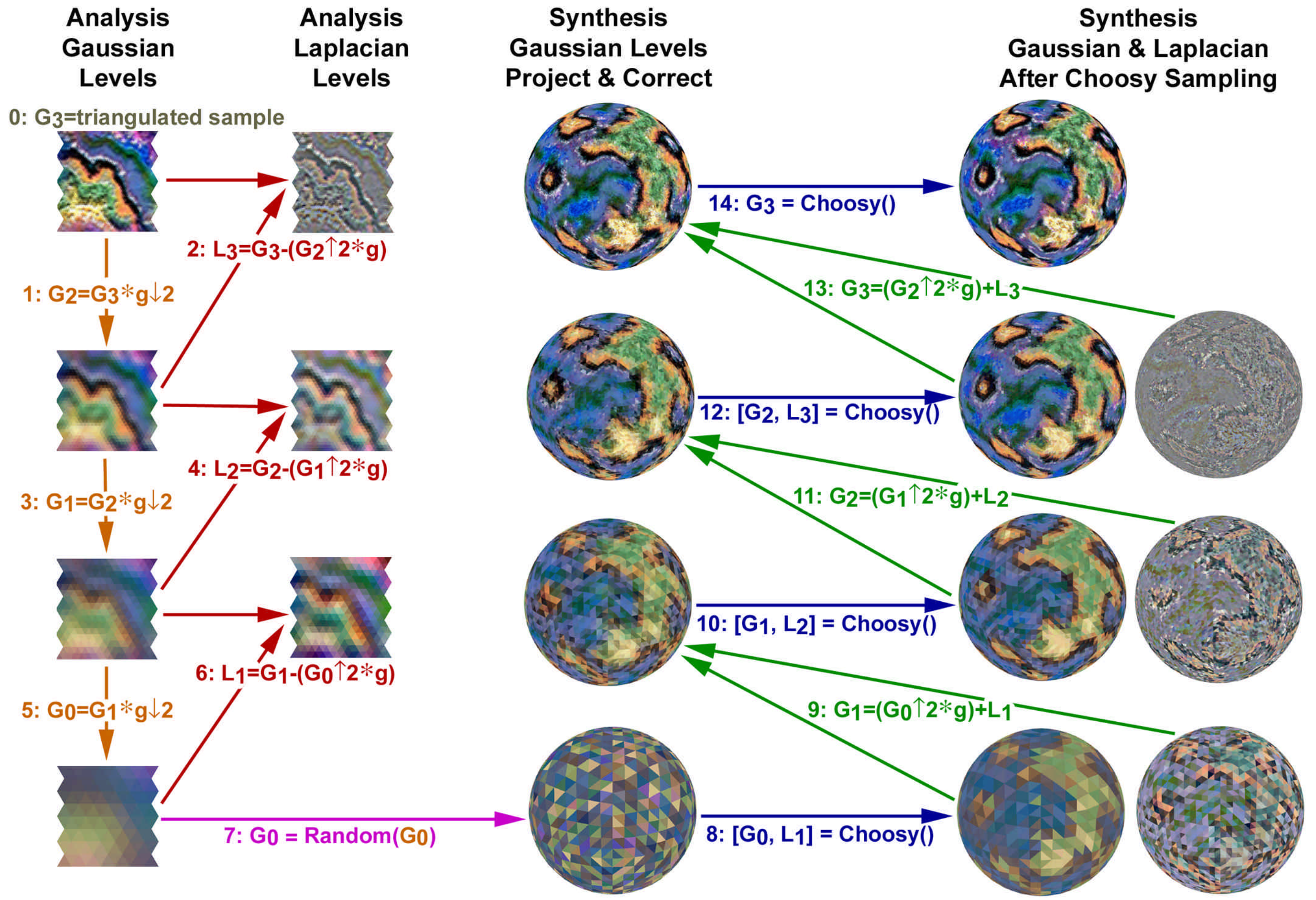

Analysis Gaussian Levels
Analysis Laplacian Levels
Synthesis Gaussian Levels Project & Correct
Synthesis Gaussian & Laplacian After Choosy Sampling
0: G_3=triangulated sample
1: $G_2=G_3*g\downarrow2$
2: $L_3=G_3-(G_2\uparrow2*g)$
3: $G_1=G_2*g\downarrow2$
4: $L_2=G_2-(G_1\uparrow2*g)$
5: $G_0=G_1*g\downarrow2$
6: $L_1=G_1-(G_0\uparrow2*g)$
7: G_0 = Random(G_0)
8: [G_0, L_1] = Choosy()
9: $G_1=(G_0\uparrow2*g)+L_1$
10: [G_1, L_2] = Choosy()
11: $G_2=(G_1\uparrow2*g)+L_2$
12: [G_2, L_3] = Choosy()
13: $G_3=(G_2\uparrow2*g)+L_3$
14: G_3 = Choosy()

Fig. 3. Analysis and Synthesis: This figure illustrates the paths of information flow in our method. The analysis pyramid is computed by first mapping the texture sample onto a triangular grid (Step 0). Next, convolution with a Gaussian kernel and downsampling is repeatedly applied. Synthesis begins by initializing G_0 of the synthesis pyramid with random values from G_0 of the analysis pyramid (Step 7). Gaussian convolution is represented by $*g$. Upsampling and downsampling are represented by $\uparrow 2$ and $\downarrow 2$. Blue arrows indicate our non-parametric sampling process, which returns both a Gaussian and a Laplacian level. Green arrows indicate the *project* and *correct* operations that produce the next level's initial guess. Note: Laplacian pyramid images generally contain negative values; therefore, for the purpose of display, the Laplacian images have been normalized.

Generalizing the Laplacian pyramid to operate on a surface requires first generalizing the basic image processing operations mentioned above: Gaussian convolution, and up and down sampling. These generalizations are described below along with a method for recursive subdivision which is used to generate the geometry on which the transform is defined.

3.4 Mesh Subdivision

Butterfly subdivision developed by Zorin, Schröder, and Sweldens [14] is used recursively to generate the geometry of each subsequent level of the pyramid. Butterfly subdivision adds a vertex to every edge, and subdivides every face into four faces as shown in Fig 4.

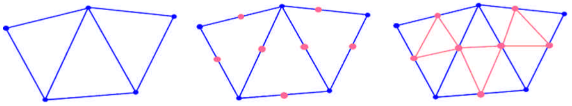


Fig. 4. Butterfly Subdivision: new vertices and edges shown in red.

The position of each new vertex is determined by the stencils shown in Fig. 5 using weights given in [14]. Vertices added to interior edges will always have a valence of six, and vertices added to boundary edges will always have a valence of four. The scheme neither deletes nor moves existing vertices, and produces limit surfaces with continuous first derivatives.

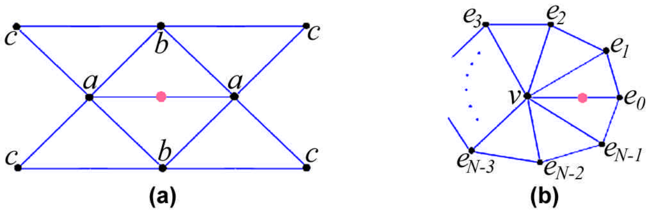


Fig. 5. Subdivision Stencils: (a) For pair of vertices each with valence=6, and (b) Used when at least one of the endpoints of the divided segment has a valence not equal to 6.

3.5 Downsampling and Upsampling

When a two-dimensional, pixel image is downsamped, by a factor of two, every other pixel in both dimensions is discarded. Upsampling, by a factor of two, inserts zeros between each pixel in both dimensions.

Sirl-Moore and Williams [10] generalized downsampling and upsampling to a surface whose faces are two-colorable by marking each face as either type *black* or type *white* so that no two adjacent faces have the same type. Recall that the butterfly scheme subdivides each parent face on level k into four child faces on level k+1. Using the method of Sirl-Moore and Williams, the center child inherits the type of its parent, and each of the three outer children inherits the type opposite to its parent (see Fig. 6).

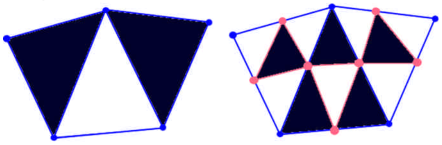


Fig. 6. Color Inheritance for Resampling

Downsampling, populates each parent cell with the RGB color values of its center child. Upsampling, populates each *white* child with the RGB color values of its parent, and each *black* child with RGB values of zero.

3.6 Gaussian Convolution

Two-dimensional, discrete convolution is a common image processing operation. To define the convolution of two functions of a surface, image pixels are replaced with surface faces. The faces, in general, are neither equally spaced nor equally sized. Therefore, the Gaussian must be represented as a continuous function of the distance, $d(i,j)$, between face i and face j. We define the distance, $d(i,j)$, between faces i and j as the straight-line Euclidean distance between the centroid of the two faces. Since the Gaussian attenuates rapidly with increasing distance, we find this to be a reasonable approximation. This is especially true at the finest resolution, which, for a 4-level pyramid, has 256 times as many faces as the input geometry. Since the butterfly subdivision produces surfaces with continuous first derivatives, even regions of the input geometry with high curvature will be reasonably flat on a small neighborhood of the finest resolution level. Differences in face size are accounted for by scaling the Gaussian by the area, a_i, of face, i.

Let $\Omega(i)$ be a neighborhood of faces about face i such that each face $j \in \Omega(i)$ is sufficiently near i to significantly contribute to the value of face i after convolution. Since the Gaussian has compact support, this will be a small neighborhood. Formally, we define the neighborhood, $\Omega(i)$, as including i, all faces adjacent to i, and all faces adjacent to those faces recursively with the recursion ending in each direction when the distance, $d(i,j)$, between the face in question and i, is greater than $3s$. The convolution weight of face $j \in \Omega(i)$ relative to face i is given by the equation developed in [10]:

$$w_{ij} = g\left(\frac{d(i,j)}{s}\right)a_j \tag{3}$$

where $g(r) = e^{-r^2/2}$, and s is the maximum distance between all pairs of adjacent faces. In the expression for w_{ij}, dividing by s dilates g, a Gaussian of unit standard deviation, so that it is a Gaussian of standard deviation s. The standard deviation of the Gaussian at each level of the Laplacian pyramid therefore equals the maximum distance between any two adjacent faces of the triangulation at that level. This is a conservative choice (compared to the alternative of setting s equal to the average distance between adjacent faces) which minimizes the aliasing introduced by downsampling, yet remains faithful to Burt's formulation [2]. The down side of this conservative choice is the risk of introducing excessive blurring. Oversampling is minimized by remeshing the input geometry (before subdividing) so that all triangles are approximately equilateral and approximately equally sized. Therefore, the maximum distance between any two adjacent faces is not much larger than the average distance. As will be discussed later, there are a number of other reasons why it is desirable to regularize the input geometry. Finally, so that the sum of the convolution weights contributing to the value of face i is one, each weight is divided by the factor, $\sum_{j\in\Omega(i)} w_{ij}$. The value of face i after convolution is therefore:

$$v_i' = \frac{\sum_{j\in\Omega(i)} (w_{ij})v_j}{\sum_{j\in\Omega(i)} w_{ij}} \tag{4}$$

where v_j is the value of face j before convolution.

3.7 Reduce and Project

With resampling and convolution defined for triangulated surfaces, *reduce* and *project* can be likewise generalized. Analogous to the case of a 2D image, *reduce* is Gaussian convolution followed by downsampling, and *project* is upsampling followed by Gaussian convolution. When a surface is upsampled, only half of the faces of the child level inherit the value of their parent. The other half of the faces are each assigned a value of zero. In order to preserve overall brightness, a normalization factor is applied to each face. This factor is developed in [10]:

$$\Gamma_i = \sum_{j\in\Omega(i)} S(j)w_{ij} \tag{5}$$

where $S(j)$ equals one if face j is *white* (meaning it inherited RGB color values from its parent), and $S(j)$ equals zero if face j is *black*. The value of face i after upsampling and convolution is therefore:

$$v_i' = \sum_{j\in\Omega(i)} v_j\left(\frac{w_{ij}}{\Gamma_i}\right). \tag{6}$$

3.8 Mapping Rectangular to Triangular Grids

What we call the *analysis pyramid* (*i.e.* the pyramid representation of the sample image), actually consists of two pyramids. The sample image, represented as a rectangular array of pixels, is mapped onto a pair of triangular "grids". The grids are identical except that the second is rotated 90° relative to the first. Each mapping forms the finest resolution level of a Gaussian pyramid representation of the sample image. During texture synthesis, when evaluating neighborhood similarity, we query both Gaussian pyramids and, within each pyramid, consider three, 60° rotations of each neighborhood.

We therefore choose the best of six rotational phases of the sample: 0°, 30°, 60°, 90°, 120°, and 150°. This effectively increases the number of sample neighborhoods by a factor six and is tantamount to averaging the Markov random field statistics over multiple orientations. This is consistent with our assumption of isotropy.

Each triangular grid consists of rows of congruent, equilateral triangles. Let K be the desired number of pyramid levels. The sample image is assumed to be square, having a width of m pixels where m is divisible by 2^{K-1}. The sample image is mapped to a triangular grid with m rows, and a second triangular grid with m columns. Since these grids correspond to the geometry of the finest resolution level of the synthesis pyramid (*i.e.*, the pyramid representation of the texture being synthesized), we must construct them so that they are the result of K-1 applications of butterfly subdivision. Fig. 7 shows the grid, G_2, used to map a 12×12 pixel sample image with K=3. G_2 has 12 rows of triangles: one row for each pixel. Notice that this geometry is the result of two subdivisions of G_0. G_0 is constructed with $m/2^{(K-1)} = 3$ rows of triangles. The number of triangles in each row of G_0 is chosen so that the width of the row is less than or equal to the total height of G_0.

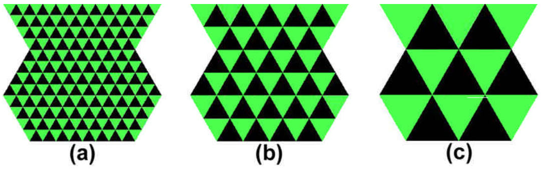


Fig. 7. Sample Geometry Mapping: (a) G_2, (b) G_1, and (c) G_0 for a 12x12 pixel sample image with N=3.

Because there is no one-to-one mapping between the rectangular grid of pixels and the grid of equilateral triangles, bilinear interpolation is used to perform the mapping. Fig. 8 shows the mapping of a 64×64 pixel texture onto 0° and 90° rotations of the flat, regular, triangular grid. The triangular grid used in this mapping consists of 64 rows with an average of 104 triangles per row. It results from four subdivisions of a roughly square grid consisting of 8 rows of congruent, equilateral triangles. The difference between the two mappings requires magnification to discern. Yet, as can be seen in the magnified images, there are significant differences in the neighborhoods. For example, compare the short, black horizontal line that appears just above the center of the magnified sample (Fig. 8d) with the same line's appearance in Fig. 8e and Fig. 8f.

3.9 Surface Remeshing

There are at least three reasons why it is desirable to remesh the input surface: First, faces of the triangulation are used in a way that is analogous to the pixels of an image. Therefore, it is desirable that all faces have approximately equal area, and be approximately equilateral. Second, the downsampling and upsampling processes, which we use for subdivision surfaces, assume that the triangulation is two-colorable. A conformal, triangulated surface is two-colorable if and only if all of its non-boundary vertices are of even valence (boundary vertices may be of any valence). Finally, computing the error distance between neighborhoods of the input and output Gaussian pyramids, as required by our sampling procedure, is most accurate when all of the vertices have a valence of six. This will be the case for vertices created by recursive subdivision, but not necessarily for those of the input geometry.

The remeshing method given by Turk [11] can be used to convert a polygonal mesh representing the geometry of a surface into a semi-regular triangulation with a user specified number of vertices. Furthermore, it can be arranged so that all vertices are even, and have a valence of six. The initial polygonal surface may be either concave or convex, and it may have holes. The major restriction is that the mesh must be conforming: Each edge is shared by no more than two polygons. In our implementation, the given triangulation is remeshed using [11]. The faces of the remeshed triangulation form the geometry of the residual lowpass level, G0, of the synthesis Gaussian pyramid.

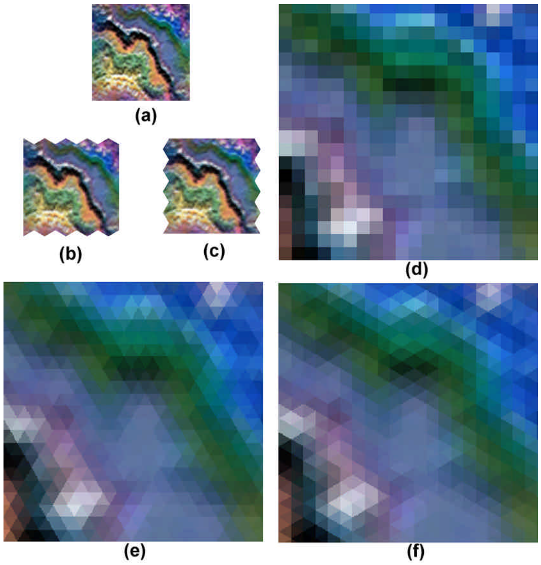


Fig. 8. Interpolation of pixel grid onto triangular grid: (a) Original texture image consisting of 64×64 = 4096 pixels. (b) Interpolated image onto a triangular grid consisting of 6,656 equilateral triangles. (c) Interpolated image onto a triangular grid that is rotated 90° from the grid used in (b). (d - f) Enlargements of corresponding segments of (a), (b), and (c).

3.10 Convolution Parameters

After building the analysis and synthesis geometries, and computing the Laplacian pyramid transform of the input texture, there remain a few parameters that need to be calculated before the synthesis process can begin. First, a scaling factor must be determined relating distances in the analysis and synthesis spaces. Distances in the analysis pyramid are left unscaled. All distances measured in the synthesis pyramid are scaled by the factor, $\hat{d}_A(k)/\hat{d}_S(k)$, where $\hat{d}_A(k)$ is the mean Euclidean distance between centroids of adjacent faces of level k, of the analysis pyramid, and $\hat{d}_S(k)$ is the corresponding quantity for the synthesis pyramid. Recall that, by construction, the distances between centroids of all pairs of faces on the analysis pyramid are identical, and on the synthesis pyramid the variance of this distance is reasonably small.

The second parameter that needs to be calculated is s_k, the standard deviation of the Gaussian kernel used in the *reduce* and *project* operations. Since values from the analysis pyramid are copied into the synthesis pyramid, this parameter should be the same for both pyramids. We set s_k equal to the maximum distance (scaled as described above) between pairs of adjacent faces for a given level of the synthesis pyramid.

3.11 Choosy Sampling

We have developed a sampling procedure for Markov random fields, which we call *choosy sampling.* This procedure finds a match based on neighborhood similarity, thereby matching the (second-order statistics of the sample, while simultaneously preserving the first-order statistics. The choosy sampling procedure, like the procedures employed by both Efros-Leung[6] and Wei-Levoy[13], is non-parametric. Non-parametric sampling, is sampling that makes no reliance on the estimation of parameters such as mean, standard deviation, or distribution curve.

Choosy sampling visits, in random order, each face of a given level of the synthetic Gaussian pyramid and asynchronously updates its color values. A similarity metric is calculated between the color values in a neighborhood of faces around the visited synthesis face and the neighborhood around each face of the corresponding level of the analysis Gaussian pyramids. The similarity metric is multiplied by a penalty factor that increases exponentially with the differential frequency of selection. More specifically, the similarity between a neighborhood, $\Omega_S(i)$, around face i of the synthesis Gaussian pyramid, and a neighborhood, $\Omega_A(j)$, around face j of the analysis Gaussian pyramid, is scaled by the factor $2^{(c[j]-c_{min})}$, where $c[j]$ is the number of times that face j already been used, and c_{min} is the minimum value of c among all neighborhoods of the appropriate level of the analysis Gaussian pyramid. This exponential scaling is a soft constraint, which is intended to approximate sampling *without replacement.* In true sampling without replacement, before any face can be used n+1 times, all faces must be used n times. Although it is clear that such a process will match the first-order statistics perfectly, true sampling without replacement has the serious problem that when a majority of the faces have been chosen n times, the search process becomes over-constrained, and it is nearly impossible to find a satisfactory match. We have observed that this results in noticeable speckling in the texture being synthesized. By approximating sampling without replacement using a soft constraint, our choosy sampling avoids this problem while still yielding good matching of first-order statistics.

By contrast, Wei-Levoy[13] always chose the best match. We have observed (see section 5) that, for many textures, this greedy selection method results in some faces (or pixels) being chosen significantly more frequently than others. The result, with such textures, is poor matching of the first-order statistics.

Efros-Leung[6] type sampling methods choose randomly between faces (or pixels) with neighborhood similarities below a given threshold. Tight thresholds capture second-order statistics at the expense of first-order statistics. The reverse is true for loose thresholds. The “best” threshold choice is texture dependent.

3.12 Neighborhood Similarity Function

The neighborhood we use in the choosy sampling is restricted to a single level of the Gaussian pyramid. The k-distant neighborhood about face j is defined as the set of faces whose minimum adjacency distance is greater than zero and less than or equal to k. If every vertex has a valence of six, then the adjacency relationships between faces in the k-distant neighborhood will always have three-fold symmetry, and the cardinality of the neighborhood will be divisible by three. The stencil shown in Fig. 9 highlights the three-fold symmetry for 2-distant neighbors:

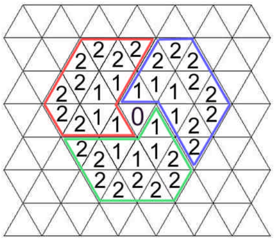


Fig. 9. Choosy Neighborhood Stencil Showing Three-fold Symmetry: Notice that there are 12 one-distant neighbors, and 24 two-distant neighbors.

In both the analysis Gaussian pyramid (by initial construction, and butterfly subdivision), and the synthesis Gaussian pyramid (by remeshing, and butterfly subdivision) the all vertices have a valence of six except for those on the boundary, which have a valance less than six. Analysis neighborhoods that include boundary vertices are not included in the sampling process. Faces missing from a synthesis neighborhood because the neighborhood extends beyond the boundary are treated as “wild cards”. The color difference between such faces and any analysis face in the corresponding position is defined as zero.

As shown in Fig. 9, a two-distant neighborhood includes $3p = 3 \cdot 12 = 36$ faces. Since each face has three, color values (RGB), there are a total of $3 \cdot 36 = 108$ values per neighborhood. The error distance between the query neighborhood $\Omega_S(i)$ of the synthesis pyramid about face i and a neighborhood $\Omega_A(j)$ of the analysis pyramid about face j, is the 108 dimensional Euclidean distance between $\Omega_S(i)$ and $\Omega_A(j)$. Since we assume isotropic textures, we exploit the three-fold symmetry of the neighborhood, by defining the error distance between $\Omega_S(i)$ and $\Omega_A(j)$ as the minimum error distance between $\Omega_A(j)$ and three 60° rotations of $\Omega_S(i)$.

Proximity is emphasized by weighting the error distances between RGB values of corresponding faces $z_S \in \Omega_S(i)$ and $z_A \in \Omega_A(j)$ by an exponential function of the inverse Euclidean distance between the centroid of each face in the syntheses neighborhood z_S, and centroid of i. Since this is the same distance function used previously for other purposes, we denote it with the same notation, $d(z_S, i)$. The scaling factor, b_Z, is given by:

$$b_z = e^{-\frac{1}{2}\left[\frac{d(z_S,i)}{1.3 s_k}\right]^2} \tag{7}$$

where, σ_k is the standard deviation of the Gaussian kernel used in the *reduce* and *project* operations at level k. So that the same numerical value of σ_k used in *reduce* and *project*, $d(z_S, i)$ must be scaled by the ratio $\hat{d}_A(k)/\hat{d}_S(k)$ defined earlier. The effect of the scaling factor, b_Z, on a relatively flat patch of geometry with approximately congruent, equilateral triangles of average size is to weight one-distant neighbors roughly four times more heavily than two-distant neighbors.

3.13 Sampling Iteration

The texture synthesis results can be improved by iteratively applying the choosy sampling at each resolution level. The improvement is particularly noticeable on samples that are composed of meaningful elements (such as the olives in Fig. 10). Successive iterations visit faces in different random orders.

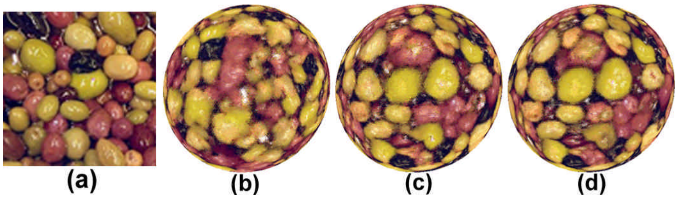


Fig. 10. Choosy Sampling Iteration on Wet Olives: (a) Sample (96×96). (b) Result with one iteration per resolution level. (c) Result with 8 iterations on levels 0 through 2, and one iteration on level 3. (d) Result with 8 iterations on each level.

4 ALGORITHMS

Pseudocode of the proposed texture synthesis process is shown in Algorithm 1 and 2. The algorithms are preceded by a list of definitions.

Δ_{input}: Geometry (faces, edges, and vertices) of the input.

Δ_A, Δ_S: Geometry of analysis and synthesis pyramids including space for RGB values of the Gaussian and Laplacian levels.

G_A, G_S: Gaussian values of analysis and synthesis pyramids.

L_A, L_S: Laplacian values of analysis and synthesis pyramids.

TurkRemesh: Converts a general conformal mesh into a semi-regular triangulation using [11].

ButterflySubdivision: Performs recursive subdivision using the butterfly subdivision schema [14].

GetDistanceScalingFactor: Calculates scaling factor for distances measured in a given level of the synthetic geometry.

GetSigma: Calculates the target standard deviation, σ_k, of the Gaussian function used for convolution at a given resolution level for both analysis and synthesis.

RandomFaces: Populates the RGB Gaussian values of the first parameter with randomly sampled faces from the second.

Project: Performs upsampling and Gaussian convolution of the given triangulation.

Choosy: Uses the choosy sampling process (see algorithm 2), to asynchronously copy RGB values of faces from $G_A[k]$ to $G_S[k]$. Additionally, if $k < MAXLEVEL$-1, then the Laplacian RGB values of each face's four children are also copied from $L_A[k+1]$ into $L_S[k+1]$.

Minimum: Calculates the minimum value of a vector.

Neighborhood: Returns an ordered, single resolution, neighborhood in the given geometry about the given face.

ErrorDistance: Returns the minimum error distance (scaled by b_Z) of three rotational phases between the Gaussian RGB values of corresponding faces of the given neighborhoods.

ChildernOf: Given a face in level k, this function returns face indexes, in level k+1, of the children of the given face.

Algorithm 1: Synthesize Texture

```
SynthesizeTexture( $G_A$, $L_A$, $\Delta_{input}$ )
  $\Delta_S[0] \leftarrow$ TurkRemesh( $\Delta_{input}$ )
  for $k = 0$ to MAXLEVEL-1 do
    if ($k > 0$) then $\Delta_S[k] \leftarrow$ ButterflySubdivision( $\Delta_S[k-1]$ )
    $\lambda_S[k] \leftarrow$ GetDistanceScalingFactor( $\Delta_A[k]$, $\Delta_S[k]$)
    $\sigma[k] \leftarrow$ GetSigma( $\Delta_A[k]$, $\Delta_S[k]$)
  end
  $G_S[0] \leftarrow$ RandomFaces($G_A[0]$, $\Delta_S$)
  $\langle G_S[0], L_S[1]\rangle \leftarrow$ Choosy($G_S[0]$, $\Delta_S[1]$, $G_A[0]$, $L_A[1]$, 0)
  for $k = 1$ to MAXLEVEL-1 do
    for $c = \{RED, GREEN, BLUE\}$ do
      $G_S[k][c] \leftarrow L_S[k][c]$ + Project( $G_S[k-1][c]$, $\lambda_S[k]$, $\sigma[k]$ )
    end
    $\langle G_S[k],L_S[k+1]\rangle \leftarrow$Choosy($G_S[k]$,$\Delta_S[k+1]$,$G_A[k]$,$L_A[k+1]$,$k$)
  end
  return $G_S[MAXLEVEL-1]$
end
```

Algorithm 2: Choosy Sampling

```
Choosy( $G_S[k]$, $\Delta_S[k+1]$, $G_A[k]$, $L_A[k+1]$, $k$)
  $c[\ \{ j \in G_A[k] \}\ ] = 0$
  foreach ($i \in G_S[k]$ selected randomly) do
    $c_{min}$ = Minimum( $c[]$ )
    $d_{min}$ = MAXFLOAT
    $\Omega_S \leftarrow$ Neighborhood( $G_S[k]$, $i$ )
    foreach ( $j \in G_A[k]$ ) do
      $\Omega_A \leftarrow$ Neighborhood( $G_A[k]$, $j$ )
      $d$ = ErrorDistance( $\Omega_A$, $\Omega_S$ ) $* \ 2^{(c[j]-c_{min})}$
      if ($d < d_{min}$ ) then { $j_{opt} = j$; $d_{min} = d$ }
    end
    $G_S[k][i] \leftarrow G_A[k][j_{opt}]$
    if ( $k <$ MAXLEVEL-1 ) then
      $L_S[k+1]$[ChildernOf( $G_S[k][i]$ ) ] $\leftarrow$
                              $L_A[k+1]$[ChildernOf( $G_A[k][j_{opt}]$ ]
    end
    $c[j_{opt}] = c[j_{opt}] + 1$
  end
```

5 RESULTS

All of the experimental results shown in this section use four levels of resolution, one iteration of choosy sampling at each level, and a neighborhood size of 36. Results from other researchers were downloaded from the named researchers' websites. Fig. 2, Fig. 11, and Fig. 12 show natural textures on a variety of geometries. OpenGL lighting effects have been used in these three figures to accentuate the surface geometry. While lighting effects can aid in realism, they can obfuscate some details of the texture synthesis results. Therefore, most of our examples (especially the comparative examples) are synthesized on the surface of a sphere without the use of lighting.

The rusty metal texture synthesized on the teapot (Fig. 11) is particularly challenging to many synthesis methods because it contains fine scale pitting, and a single instance of a large-scale rusty patch. This patch has a very distinctive shape, which would be distracting if reproduced exactly in different places on the surface, as quilting methods are prone to do.

Burl wood (Fig. 12) is one of the few wood grains that is statistically isotropic.

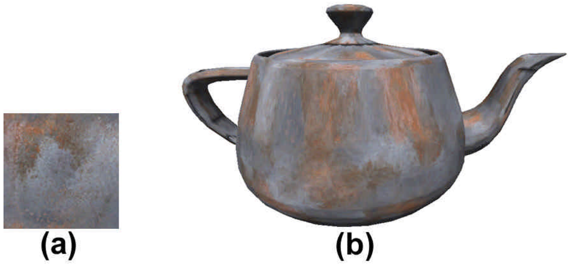


Fig. 11. Rusty Metal texture on Utah Teapot: (a) Samples (96×96), (b) 3,752 face input geometry and 960,521 face output geometry displayed with OpenGL lighting effects.

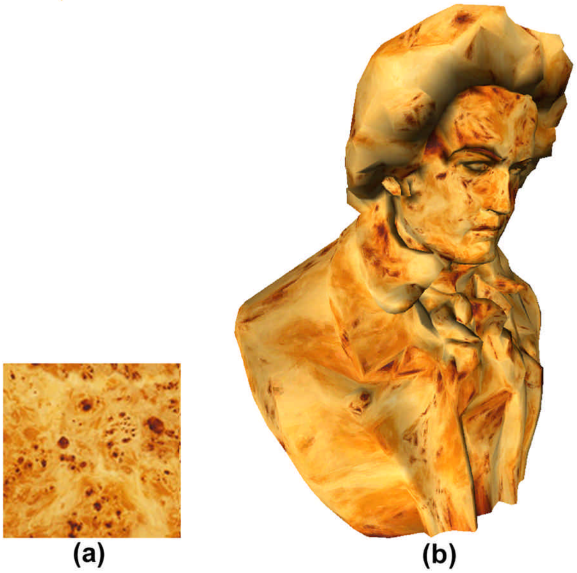


Fig. 12. Burl Wood Synthesized on Ludwig von Beethoven Bust - (a) Sample (128×128), (b) 5,030-face input geometry and 321,920-face output geometry displayed with OpenGL lighting effects.

The candy sample shown in Fig. 13 is composed of repeated elements (roughly cylindrical candies). While the texture as a whole is isotropic, the elements from which it is comprised are not. Our method often fails to reproduce the elements of such a texture in an accurate way (although the results can be significantly improved with multiple iterations of choosy sampling. See Fig. 10). However, we believe that we do better with this candy texture than Wei-Levoy[13]: First, the shape of our candies more closely matches those in the sample. Secondly, the first order statistics of Wei-Levoy's result are severely skewed. Their result contains too much lavender and brown, and too little green and teal. This point is dramatically illustrated by the histograms. Our results match the sample's histograms so closely that they are essentially superimposed. As explained earlier, the Wei-Levoy[13] sampling method is too greedy and often results in poor matching of the first-order statistics. The histograms in Fig. 13 were calculated with 50 bins in each color channel. The sample contains 36,864 pixels, and our result contains 131,072 faces (counting the full surface of the sphere). The result from Wei-Levoy contains 40,000 pixels. Each histogram has been normalized so that the sum of all bin-counts equals is one.

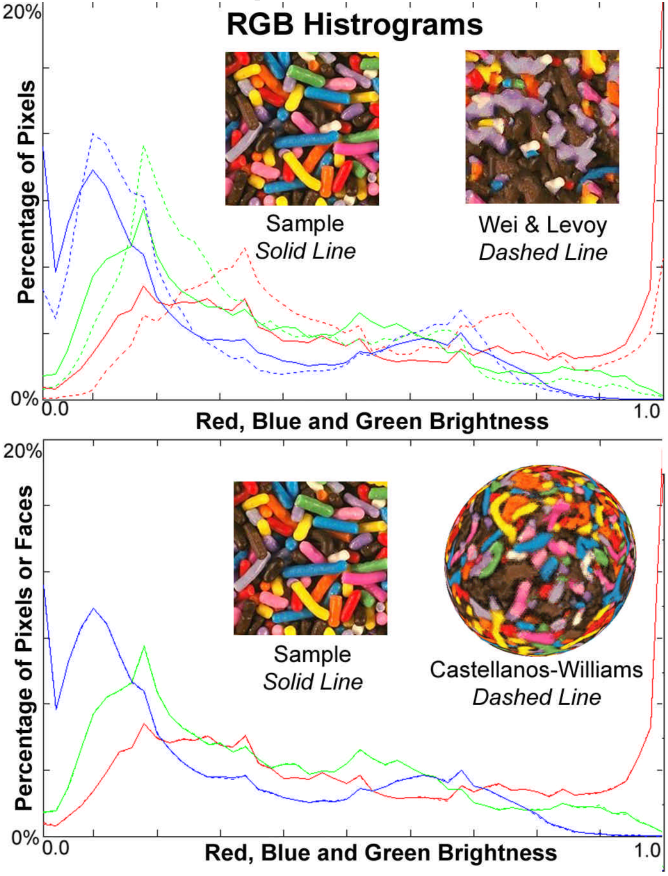


Fig. 13. Histogram Comparison for Candy Texture.

The pool water, (Fig. 14), works exceptionally well with our method. We succeed in capturing the first order statistics, and the connectedness, thickness, and randomness of the lines of light, as well as the shading of the darker areas. The synthetic patterns are novel, yet realistic.

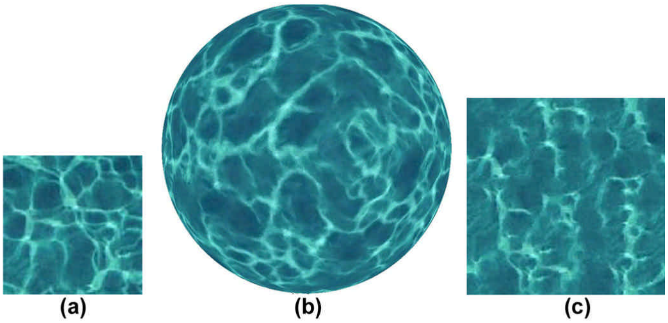


Fig. 14. Pool Water: (a) Sample (128×128), (b) Castellanos-Williams on 524,288 face sphere, (c) Wei-Levoy

The textures in Fig. 15 and 16 are each anisotropic on a medium scale. For instance, in Fig. 15, the black bands form a roughly rectangular grid. This anisotropy is well modeled by the method of Wei-Levoy[13]. Our method creates isotropic variants of anisotropic samples. Although these are not faithful reproductions, they are often quite interesting. Unlike the isotropic textures they are derived from, they can be synthesized directly on a two-dimensional surface in the absence of a parameterization. Also note that Wei-Levoy[13] succeed in matching the first order statistics for each of these textures. We believe that the reason for this is that in each case, the first-order statistics are fairly constant in all small windows of the sample.

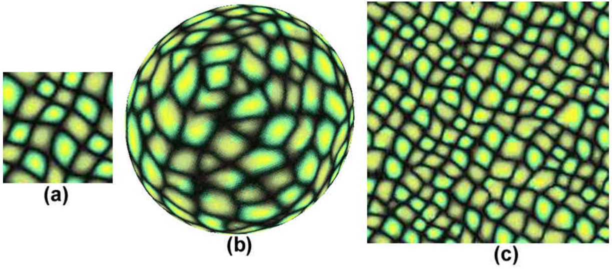


Fig. 15. Alien Skin Comparison: (a) Sample (96×96), (b) Castellanos-Williams on 131,072 face sphere, (c) Wei-Levoy

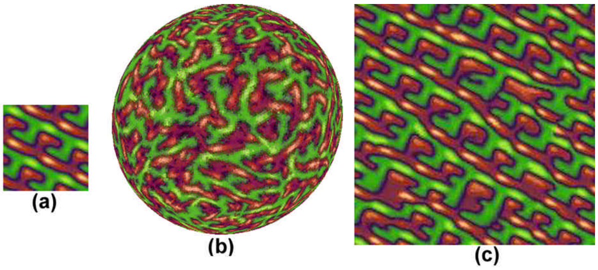


Fig. 16. Anisotropic Red-Green Pattern Comparison: (a) Sample (64×64), (b) Castellanos-Williams on 131,072 face sphere, (c) Wei-Levoy.

The sample used in Fig. 17 is a segment from an abstract watercolor painting. Our method has created completely novel patterns (no copying) within the artist's style.

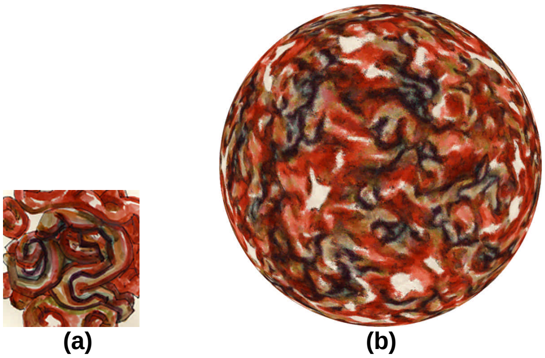


Fig. 17. (a) Isotropic Watercolor (96×96), (b) Castellanos-Williams.

The saltine cracker sample, shown in Fig. 18, offers an excellent comparison of the strengths and weakness of the various methods. This sample contains a deterministic pattern of holes superimposed on an isotropic background texture. The quilting method used by Efros-Freeman[5] reproduces the deterministic pattern perfectly. Notice also that there are no discernible boundaries between the patches. Efros-Freeman[5] capture the large-scale stochastic pattern of the placement of the dark brown patches. The smaller scale stochastic properties, however, are poorly represented. In particular, the same dark random shape is reproduced distractingly throughout the image. Wei-Levoy[13] also show evidence of copying what should be a randomly shaped dark spot. Additionally, Wei-Levoy[13] produce far too much light colored area. Our method, while it completely fails to reproduce the arrangement of the holes, creates genuinely novel brown patches that resemble those of real crackers. It also accurately reproduces the sample's color mixture.

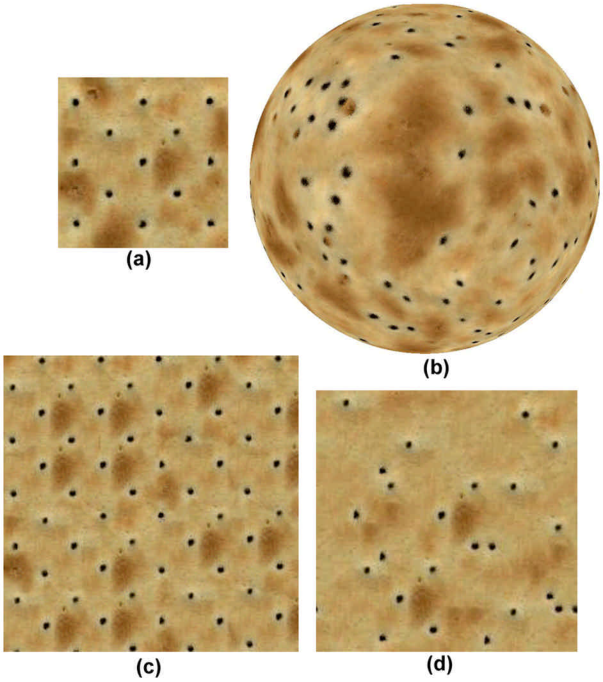


Figure 18 - Saltine cracker Comparison: (a) Sample, (b) Castellanos-Williams on 524,288 face sphere, (c) Efros-Freeman, (d) Wei-Levoy.

6 ACCELERATION

6.1 Time Complexity Without Acceleration

The most computationally intensive part of the algorithm described above is the choosy sampling. We usually use a neighborhood size of 36 with three colors. Queries, therefore, are performed in d-dimensional space where $d = 3\times36 = 108$. The time complexity with M faces on the final rendered surface, and N pixels in the sample is, therefore,

$$\sum_{k=0}^{K} O\left(6MNd/2^{k}\right) = O(MNd) \qquad (8)$$

The K terms correspond to the number of pyramid levels (we usually use four), and the factor of six comes from the use of six rotational phases of the sample.

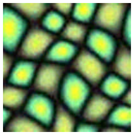
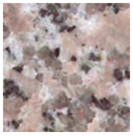
6.2 Dimensionality Reduction

The Eigenvector Transform, variously referred to as Karhunen-Loéve (KL) Transform, Hotelling Transform, or Principal Component Analysis (PCA) is a procedure that transforms a number of correlated random variables into a smaller number of uncorrelated random variables called the *principal components* [8]. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. Given N vectors, $\vec{x}_i$ of dimension d, the principal components are the eigenvectors of XX^{T}. Principal components with very small eigenvalues can be ignored in many problems without significantly affecting the results.

In our problem, $\vec{x}_i$ is a three-color, adjacency neighborhood totaling 108 dimensions, and N is the number of faces in a given level of the sample. Therefore, there are 108 eigenvectors and eigenvalues. For all of the textures we have tested (which include all those shown in this paper), the ratio of the first eigenvalue to the second is at least 50, and the ratio of the first to the twentieth is at least 1000. Table 1 shows five sample textures and selected eigenvalues.

TABLE 1

NEIGHBORHOOD EIGENVALUES

Selected Eigenvalues				
108	0.00000	0.00000	0.00003	0.00000
75	0.00007	0.00003	0.00041	0.00001
50	0.00036	0.00030	0.00103	0.00007
30	0.00379	0.00105	0.00542	0.00081
20	0.01256	0.00375	0.01743	0.00270
10	0.04825	0.02663	0.14606	0.01431
9	0.05045	0.03131	0.17085	0.01533
8	0.08063	0.03761	0.17469	0.01823
7	0.10439	0.03973	0.28112	0.01975
6	0.36311	0.10023	0.30249	0.04339
5	0.37680	0.11087	0.39283	0.04381
4	0.45952	0.11649	0.56957	0.04832
3	1.27433	0.39757	0.60274	0.14290
2	1.28836	0.42776	1.52357	0.15274
1	72.5694	176.469	74.2641	81.1529

Four sample textures and their eigenvalues for 108´108 covariance matrix.

Recall that in our procedure, proximity is emphasized by weighting the error distances between RGB values of corresponding faces by a Gaussian function of the faces' Euclidean distance to the face being synthesized. In the transformed space, each dimension no longer corresponds to a single face making it impossible to apply distance weighting. Therefore, weighting needs to be applied before the neighborhood is transformed. Then, at each query, the same scaling needs to be applied to the magnitudes of the synthesized neighborhood values.

The images in Fig. 19 show a comparison of results using the full 108 dimensional query and results using only the 20 most significant transformed dimensions. We observe that the quality of the results is indistinguishable. The overhead added by the transformation is negligible, and this process immediately offers a five-fold increase in speed. More significantly, the dimensionality reduction makes it possible to use specialized data structures for approximate nearest neighbor computations in high-dimensional spaces, such as Kd-trees, leading to further acceleration.

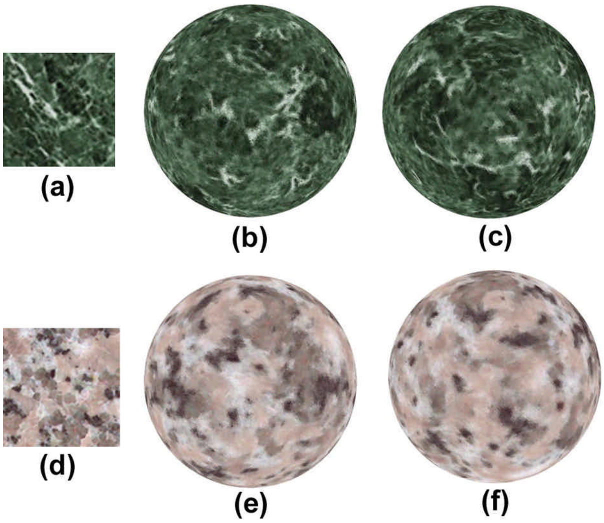


Fig. 19. Acceleration Comparison: (*a*) Green Marble Sample (96×96). (b) Orthoclase Garnet sample (96×96). (*b*, *e*) Synthetic textures using, 108-dimention query without distance weights. (*c*, *f*) Result with 20-dimention query in transformed space.

6.3 Kd-Tree of Transformed Neighborhoods

A Kd-tree (k-dimensional tree) is a standard data structure for quickly searching a multi-dimensional space for an approximate nearest neighbor. There are a number of variants of the Kd-tree. The one most relevant to our needs is briefly described.

Given a set of k-dimensional points, a Kd-tree is constructed recursively by choosing the dimension with the greatest variance, calculating the median value in that dimension, and splitting the set roughly in half. Points along the selected dimension with values less than the median are placed in one subtree, and the remaining points are placed in the other subtree. This is done recursively until each subtree contains only a single point. A Kd-tree for a set of n points uses $O(n)$ storage, can be constructed in $O\left(kn\log_2 n\right)$ time, and can be queried for an approximate nearest neighbor in $O\left(k\log_2 n\right)$ time. This search works well for a wide range of data sets provided n is much larger than k.

We have seen that by using the eigenvector transform, the dimensionality of neighborhood searches can be reduced from 108 to 20 without a significant loss of information. Our data sets consist of 8,000 points (for the finest resolution level of a 64×64 sample) through 30,000 points (for a 128×128 sample). Given an arbitrary data set, this number of points is marginally sufficient for a twenty dimensional Kd-tree. However, as made explicit by the eigenvector transformation, the variance of our dataset is concentrated in the principle components with the largest eigenvalues; therefore the tree will, on average, split more frequently along dimension i than along the dimension i+1. Recall that in our particular data sets, the variance along the first dimension was found to be at least 50 times greater than the variance along the second dimension. This concentration of the variance insures that the tree is maximally effective for the limited dataset.

Using the eigenvector transform described earlier, each leaf in the Kd-tree is a transformed representation of the RGB values of a neighborhood of 36 Gaussian faces in one of the two analysis pyramids. A query point consists of the transformed RGB values of a neighborhood of Gaussian faces in the synthesis pyramid.

6.4 Set of Dynamic Kd-Trees

The requirement for approximately uniform sampling makes the use of a standard Kd-tree impossible. A Kd-tree can be queried to find an analysis face with a neighborhood similar to a given neighborhood. The choosy sampling procedure requires that the difference between the given neighborhood and the query result be weighted by a function of the number of times that result has been chosen. This weight cannot be built into the tree from the start because it changes after every query and is a function of the query point as well as the leaf.

We propose the use of a set of dynamic Kd-trees: A single Kd-tree could initially hold all of the neighborhoods of the sample. When a leaf is chosen, the chosen leaf needs to be moved to a different tree, so that all nodes in a given tree have been chosen the same number of times. A search then, would query each tree individually and appropriately scale the result from each tree. By the nature of the data and the exponential scaling factor, the number of leaves in each tree would form a narrow Gaussian with the vast majority of leaves in only 3 trees. It is important that only a small number of trees are needed because each tree requires a separate search, and Kd-trees only give reasonably accurate results when they have many more leaves than dimensions. Points that do not fall within the three most populous values of $c[j]$ could be searched linearly. Thus, the complete texture synthesis algorithm is improved from $O(mnk)$ to $O(mk \log n)$.

7 CONCLUSION

We have described a fully automatic method of synthesizing isotropic textures on subdivision surfaces from sample images. Within the class of isotropic textures, this method offers several improvements over existing methods. The method can also be used to create isotropic variants of anisotropic textures.

The original contributions of our paper can be summarized as follows:

- Both Gaussian and Laplacian pyramid representations of the sample texture are constructed. The inverse Laplacian pyramid transform is used to generate first approximations to the texture at each level of the associated Gaussian pyramid resulting in a smooth coarse-to-fine information flow.
- We have introduced a new non-parametric sampling method, which approximates sampling without replacement, yet is softly constrained. Choosy sampling transforms a greedy, nearest neighbor search process into a significantly more effective non-parametric sampling method. The essential property of searching for a best match (and in doing do, modeling a texture's second-order statistics) is maintained while simultaneously preserving the first-order statistics of the sample texture.
- The input texture is resampled using a triangular grid so that its geometry more closely resembles that of the subdivision surface on which the texture will be synthesized. Representing both the analysis and synthesis pyramids using the same data structure greatly simplifies the neighborhood matching procedure that is the heart of the texture synthesis process.
- The application of the Eigenvector Transform to the set of all sample neighborhoods at a given resolution level has proven highly effective in accelerating the texture synthesis process.
- The standard Kd-tree has been adapted to apply to a case where the result of a query is a function of the result of all previous queries.

References

[1] Bar-Joseph Z., El-Yaniv R., Lichinski D., and Werman M. “Texture Mixing and Texture Movie Synthesis using Statistical Learning,”. *IEEE Trans. on Visualization and Computer Graphics* vol. 7, no. 2, pp. 120–135, 2001.

[2] Burt P.J. “Fast Filter Transforms for Image Processing.” *The Visual Computer* vol. 16, pp. 20–51, 1981.

[3] Burt P.J., Adelson E.H. “The Laplacian Pyramid as a Compact Image Code,” *IEEE Trans*, C-31:532-540, 1983.

[4] De Bonet J.S. “Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images,”. *SIGGRAPH 97*, pp. 361–368, 1997.

[5] Efros A. and Freeman W. “Image Quilting for Texture Synthesis and Transfer,” *SIGGRAPH 01*, pp. 341–346, 2001.

[6] Efros A. and Leung T. “Texture Synthesis by Non-parametric Sampling,” *Intl. Conf. on Computer Vision*, vol. 2, pp. 1033–1038, 1999.

[7] Heeger D.J. and Bergen J.R. “Pyramid-based Texture Analysis/Synthesis,” *SIGGRAPH 95*, pp. 229–238, 1995.

[8] Hotelling H. “Analysis of a Complex of Statistical Variables into Principal Components,” *J. Educ. Psychol.*, vol. 24, pp. 417-441, 498-520, 1933.

[9] Praun E., Finkelstein A. and Hoppe H. “Lapped Textures,” *SIGGRAPH 00*, pp. 465–470, 2000.

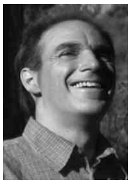
[10] Sirl-Moore Curtis and Williams Lance R. “Automatic Synthesis of Isotropic Textures on the Sphere from Sample Images Using the Laplacian Pyramid Transform,” *University of New Mexico*, TR-CS-2001-11, 2001.

[11] Turk G. “Re-tiling Polygonal Surfaces,” *SIGGRAPH 92*, pp. 55–64, 1992.

[12] Wei L.Y. and Levoy M. “Fast Texture Synthesis using Tree Structured Vector Quantization,”. *SIGGRAPH 00*, pp. 479–488, 2000.

[13] Wei L.Y. and Levoy M. “Texture Synthesis Over Arbitrary Manifold Surfaces,” *SIGGRAPH 01*, pp. 355–360, 2001.

[14] Zorin D., Schröder P. and Sweldens W. “Interpolating Subdivision for Meshes with Arbitrary Topology,” *SIGGRAPH 96*, pp. 189–192, 1996.



Joel Castellanos received his BS degree in Physics and Computer Science from Wilkes University and is currently a graduate student in the Dept. of Computer Science at the University of New Mexico. He is also currently a senior software engineer with ASRC Aerospace, has taught mathematics at Central Wyoming College, Physics at Albuquerque Academy HS, and was a research scientist Rice University.

Lance R. Williams received the BS degree in computer science from the Pennsylvania State University and the MS and PhD degrees in computer science from the University of Massachusetts. He is currently an associate professor of computer science with the University of New Mexico. Prior to joining UNM, he was a postdoctoral scientist at NEC Research Institute.