Automatic Synthesis of Isotropic Textures on Subdivision Surfaces from Sample Images

Joel Castellanos* University of New Mexico

Curtis Sirl-Moore† University of New Mexico

Lance Williams‡ University of New Mexico

Abstract

We describe a fully automatic method of synthesizing isotropic textures on subdivision surfaces from sample images. We restrict ourselves to isotropic textures because only isotropic textures can be automatically generated on an arbitrary surface in the absence of a parametrization. Unlike previous approaches, texture synthesis is accomplished in a coarse-to-fine fashion by constructing both Gaussian and Laplacian pyramid representations of the synthetic texture. The inverse Laplacian pyramid transform is used to generate first approximations to the texture at each level of the associated Gaussian pyramid. These approximations are refined using a modified nearest neighbor search process which preserves the first-order statistics of the sample texture. This search process can be considered to be a sampling procedure for an implicitly defined Markov random field. 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.

Keywords: texture synthesis, laplacian pyramid, subdivision surfaces

1 Introduction

A texture can be thought of as a visual pattern that is repeated according to rules which are applied either deterministically or at random. The pattern may be completely deterministic, such as a chess board, completely random, such as snow on a television screen, 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 preferred orientation) to strongly anisotropic. An example of the former would be polished granite, while horizontal layers of sandstone would be an example of the latter. There is a strong connection between isotropy and randomness: With the trivial exception of uniform colors, deterministic textures must be anisotropic. Finally, we observe that a texture may exhibit different degrees of randomness and isotropy at different scales. See Figure 1.

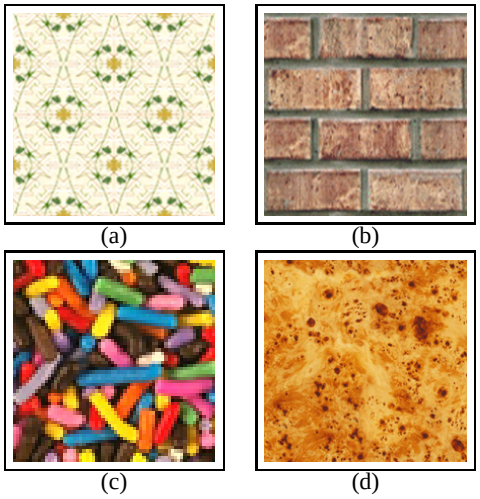

Figure 1: (a) A floral print fabric is deterministic and hence non-isotropic. (b) The large scale structure of a brick wall is non-isotropic. However, at small scales, the structure becomes isotropic. (c) Textures formed by the random arrangement of identical elements exhibit aspects of both determinism and randomness. Such textures are isotropic at all scales. (d) Some textures can only be characterized statistically. Burlwood is an example of a texture of this sort which is isotropic at all scales.

*e-mail: joel@cs.unm.edu

†e-mail: joel@cs.unm.edu

‡e-mail: williams@cs.unm.edu

Texture synthesis is the creation of a new image that is different from the original, yet can be said to have the same visual appearance as the original. Texture synthesis has been an active research topic in both computer vision and graphics. The most common application is to synthesize the texture on the surface of an object in a computer generated scene in order to increase its degree of visual realism[Ebert et al. 1994]. Other, image-based applications include hole filling[Efros and Leung 1999], image and video compression, and animation[Bar-Joseph et al. 2001].

2 Previous Work

2.1 Efros and Leung

One of the simplest and most successful texture synthesis methods is that of [Efros and Leung 1999], which was inspired by an idea of Shannon's for producing English sounding text by modeling language as a generalized Markov chain. Their method is based on the simple assumption that the probability distribution of RGB values for a given pixel only depends on the RGB values of pixels in a local neighbourhood of the given pixel, *i.e.*, the texture can be modeled as a Markov random field. Using this assumption, the texture synthesis process grows a new texture outwards from an initial seed, one pixel at a time. The value of the next pixel to be synthesized is chosen at random from the values of sample pixels with neighbourhoods resembling the neighborhood of the given pixel, *i.e.*, those with similar spatial contexts. The only user specified parameter is the size of the neighbourhood of a pixel, which controls the degree of randomness of the synthesized texture.

2.2 Wei and Levoy

Our work builds most directly upon the multi-resolution, neighborhood-search-based method developed by [Wei and Levoy 2001]. We describe their process in detail because our process parallels it. [Wei and Levoy 2001] describe methods for texture synthesis on both parameterized, and non-parameterized surfaces. However, in this paper, we are primarily concerned with their work on non-parameterized surfaces. For non-parameterized surfaces, their method takes two inputs: (1) A sample image; and (2) A triangulation of the surface on which the texture is to be synthesized. A Gaussian pyramid with K levels is built from the sample image. The triangulation of the surface is then recursively subdivided so that all the triangles are approximately equilateral and congruent. Each face of the remeshed surface is subdivided $K-1$ times. Each subdivision of the remeshed surface corresponds to a level of the Gaussian pyramid of the sample image. The original remeshed surface corresponds to the lowest level of the sample's Gaussian pyramid and the finest subdivision is the triangulation on which the final synthetic texture will be represented. The texture is synthesized in a coarse-to-fine manner. After initialization, the synthesis process takes place in two distinct matching phases per Gaussian pyramid level:

1. **Initialization.** To begin, level k is populated with randomly chosen RGB values from level k of the analysis Gaussian pyramid.

2. **Multi-level neighborhood matching.** In the first matching phase, each face at level k is visited in random order. Like [Efros and Leung 1999], a face's RGB value is replaced with the RGB value of the face from level k of the analysis Gaussian pyramid with most similar neighborhood. However, in order to propagate information from coarse-to-fine scales, the first phase neighborhood spans two levels of the Gaussian pyramid, *i.e.*, it consists of neighboring faces at level k and level $k-1$.

3. **Single-level neighborhood matching.** In the second matching phase, the final value of the synthesis face at level k is determined by repeating the non-parametric sampling procedure used in the first phase. However, unlike the first phase, which was used to propagate information from coarse-to-fine scales, the neighborhood which is used in the second phase consists solely of faces at level k in the analysis Gaussian pyramid.

Finally, in order to compare neighborhoods of the triangulated surface (on the synthesis side), with image neighborhoods (on the analysis side) they project the neighborhood of the triangulated surface onto a tangent plane and resample it using a rectangular grid.

2.3 Quilting

Quilting methods stitch together representative patches of a sample texture image. Quilting methods such as those described by [Praun et al. 2000; Efros and Freeman 2001] can be very fast and produce excellent results for a wide range of textures on parameterized surfaces. They are especially good at synthesizing deterministic textures. There are, however, at least two disadvantages inherent to any quilting process. First, quilting methods cannot reproduce structure at scales larger than the size of the quilting elements. Second, unlike the method of [Wei and Levoy 2001] (and our own method), quilting cannot be used to synthesize isotropic variants of non-isotropic textures. This limits their utility on surfaces lacking an explicit two dimensional parameterization. The reason is that error due to misalignment of quilting elements can only be minimized by exercising comparitively few degrees of freedom when compared to methods which synthesize a texure pixel by pixel.

3 Our Approach

Our method for texture synthesis on surfaces has been inspired by other multi-resolution, texture synthesis methods which use non-parametric methods to sample an implicitly defined Markov random field. Although our primary inspiration was [Wei and Levoy 2001], we have also been influenced by [Efros and Leung 1999], [Debonet 1997], and [Bar-Joseph et al. 2001]). Like [Wei and Levoy 2001] we synthesize texture in a coarse-to-fine manner by searching for the face in the analysis Gaussian pyramid with most similar neighborhood. However, our method differs from [Wei and Levoy 2001] in three significant ways. First, our method propagates information from coarse-to-fine in a more principled manner, namely, by incrementally inverting a Laplacian pyramid transform. Second, the greedy method of non-paramatric sampling described by [Wei and Levoy 2001] has been replaced with a sampling procedure of our own design which does a better job of preserving the first-order statistics of the sampled texture. Finally, unlike [Wei and Levoy 2001], as a preprocesssing 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. The consequence is that in our system, the multi-resolution pyramid data structures on both the analysis and synthesis sides are based on triangulations. This considerably simplifies the neighborhood matching procedure which is the heart of the texture synthesis process.

3.1 Coarse-to-fine Information Flow

Whereas [Wei and Levoy 2001] compute a Gaussian pyramid representation of the input texture and use it to synthesize a Gaussian pyramid representation of the output texture, our method computes both Gaussian and Laplacian pyramid representations of the input texture in order to synthesize both Gaussian and Laplacian pyramid

representations of the output texture. To this end, we have generalized the Laplacian pyramid transform so that it can be used to represent functions defined on triangulations. One property of the Laplacian pyramid which distinguishes it from the Gaussian pyramid is that, like the wavelet transforms to which it is intimately related, the Laplacian pyramid transform can be *inverted*. That is, the original function can be recovered from the Laplacian pyramid and a single residual Gaussian value. Because we synthesize both Gaussian and Laplacian pyramids, we can incrementally invert the Laplacian pyramid transform in order to propagate information from coarse scales to fine scales during the synthesis process. This mechanism avoids the awkward multi-level neighborhoods and multiple matching phases which [Wei and Levoy 2001] use to propagate information from coarse-to-fine scales.

The sampling procedure we use to sample the Markov random field at level k of the synthesis Gaussian pyramid does not just return an analysis Gaussian pyramid value, it also returns the values of the analysis Laplacian children at level $k+1$. Once the sampling process for level k of the synthesis Gaussian pyramid has converged, we initialize level $k+1$ by upsampling level k, convolving it with a Gaussian kernel, and adding this to level $k+1$ of the synthesis Laplacian pyramid. This incremental inversion of the synthesis Laplacian pyramid transform generates the first approximation of the texture at level $k+1$ of the synthesis Gaussian pyramid.

In summary, [Wei and Levoy 2001] use a two phase process at each level. Their process consists of random initialization; followed by (1) one pass of multi-level neighborhood matching; and (2) one pass of single-level neighborhood matching. We use a considerably simpler process consisting of initialization using the inverse Laplacian pyramid transform followed by one pass of single-level matching.

3.2 Sampling Procedure

The second major difference between our method and that of [Wei and Levoy 2001] is the non-parametric sampling procedure. Our sampling procedure finds a reasonably good match based on neighborhood similarity while simultaneously preserving the first-order statistics of the sample texture. This is done by using a non-parametric sampling method which approximates sampling without replacement. An example of sampling without replacement is a child repeatedly selecting and removing jellybeans from a jar. In sampling with replacement, the selected jellybean is returned to the jar before a subsequent selection is made. For finite populations, these two sampling methods give different results.

We will explain our sampling procedure using a somewhat farfetched analogy. We hope the reader will forgive us! A group of children are allowed to choose jellybeans from a jar containing many different colors. However, after choosing a jellybean, they must cut it in half and return half to the jar. A child approaches the jellybean jar with the idea of finding his/her favorite color. She searches for and selects the closest match, and then returns half of the jellybean to the jar. Subsequent selections are made in turn by different children each searching for their favorite colors. It is conceivable that for some mix of children and jellybeans that a conflict never arises: Every child always finds a whole bean of his/her favorite color. A more likely scenario is that a child occasionally only finds half beans or quarter beans of that color. In such a situation, the child may be willing to settle for a whole bean of a similar color if all beans of her favorite color have already been halved or quartered, *i.e.*, if too many children choose pink, eventually the red bean will start looking quite tasty.

Note that this sampling policy (unlike the sampling with replacement policy used by [Efros and Leung 1999] and [Wei and Levoy 2001]) tends to insure that the distribution of colors in the jar and the distribution of colors which are chosen will be the same. Furthermore, unlike sampling without replacement, the problem of respecting the choices of children who choose last is not overconstrained, *i.e.*, No child need ever choose licorice if he/she really dislikes it!

In our algorithm, this is implemented by scaling the error distance between the pair of neighborhoods being compared, *i.e.*, one in the analysis Gaussian pyramid and one in the synthesis Gaussian pyramid, by a factor of $2^{(c[j]-c_{min})}$, where $c[j]$ is the number of times that face j in the analysis Gaussian pyramid has already been selected, and c_{min} is the minimum number times that any analysis Gaussian pyramid face has been selected. When combined with the random order in which udates are performed, this policy effectively transforms the greedy heuristic used by [Wei and Levoy 2001] into a genuine sampling procedure. Furthermore, this procedure has the additional benefit of preserving the first-order statistics of the sample texture.

4 Analysis

4.1 Laplacian Pyramid Transform

The Laplacian pyramid is a linear transform that represents an image by decomposing it into a set of bandpass images, and a single lowpass image from which the original image can be perfectly reconstructed. It is a historical precursor of the fast wavelet transform.

In the first step of the Laplacian pyramid transform, the image is lowpass filtered by convolving it with a Gaussian kernel, and the result is subtracted from the original image. The high-frequency detail in the image is retained in this difference image. The lowpass filtered image (called the Gaussian pyramid level) can then be downsampled by a factor of two in each dimension without loss of information. Let G_K be the original image, and let g be a Gaussian kernel of size $2X+1$ by $2X+1$ where X is a positive integer. Then, at each step of the encoding process, the image is decomposed into a half-resolution low-frequency component, G_k, and a full-resolution, high-frequency component (called the Laplacian Level), L_k, where k is the pyramid level. More formally,

$$G_{k-1}(i,j) = \{G_k * g\}(2i,2j) \quad (1)$$

$$L_{k-1}(i,j) = G_k(i,j) - \{G_k * g\}(i,j). \quad (2)$$

Given an $m \times m$ image, then after K iterations, where $m = 2^{(K-1)}$, the residual lowpass Gaussian level, G_0, is a single pixel. The image representation consists of the set of $K-1$ Laplacian pyramid levels, L_k, and the final Gaussian pyramid level, G_0.

Image decoding is done in the reverse order. *Upsampling* is the process of inserting zeros between sample points in both dimensions. Each Gaussian pyramid level, G_k, beginning with the lowest resolution, G_0, is upsampled and interpolated by convolution with the Gaussian kernel, g. Then the result is added to the corresponding Laplacian pyramid level, L_{k+1} and the process is repeated. This process reconstructs the original image without error. Laplacian pyramid level k can be thought of as being the difference between Gaussian pyramid levels $k-1$ and k.

For notional convenience we define two operations: *reduce* and *project*. The reduce operation corresponds to convolution with a Gaussian kernel followed by downsampling by a factor of two in each dimension. The project operation upsamples by a factor of two in each dimension and then interpolates by convolution with the same Gaussian kernel. Therefore:

$$G_{k-1} = reduce(G_k) \quad (3)$$

$$L_{k-1} = G_k - project(G_k). \quad (4)$$

4.2 Surface Remeshing

Faces of the triangulations are used in a way which is analogous to the pixels of an image. Therefore, it is desirable that all faces have approximately equal area, and be approximately equilateral. The downsampling and upsampling processes which we define for subdivision surfaces additionally assume that the triangulation is two-colorable. A triangulation of a surface without boundary is two-colorable if and only if all of its vertices are of even valence. Computing the error distance between neighbohoods of the input and output Gaussian pyramids, as required by our non-parametric sampling procedure, is easiest when all of the vertices have a valence of six. This will be the case for vertices associated with faces produced by means of recursive subdivision of an input geometry represented as a triangulation.

It follows that in order to make use of existing mesh libraries it is necessary to have a method of remeshing that satisfies these constraints while preserving the original topology. [Turk 1992] describes just such a method. His method can be used to convert a polygonal mesh 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 that most have a valence of six. Finally, this process is guaranteed to preserve the topology. 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*, *i.e.*, each edge is shared by no more than two polygons.

In our implementation, the given triangulation is remeshed using Turk's method. The faces of the remeshed triangulation form the geometry of the residual lowpass level, G_0, of the Gaussian pyramid.

4.3 Mesh Subdivision

Butterfly subdivision[Zorin et al. 1996] is used recursively to generate the geometry of each subsequent level of the pyramid, G_1 through G_{K-1}. Butterfly Subdivision adds a vertex to every edge, in the process, subdividing every face into four faces. 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 is interpolating, since its neither deletes nor moves existing vertices. The scheme is triangular as it operates on triangular control nets and produces triangular limit surfaces. It is stationary as it uses the same set of rules every time it subdivides the net, and uniform because every section of the net is subdivided with the same set of rules. The rules for adding vertices are simple and have compact support.

The butterfly scheme produces surfaces with continuous first derivatives. A drawback of the scheme is that, like many interpolating schemes, it produces slight ripples and undulations in the surface, especially near tightly joint areas. These ripples show up most clearly on monochromatic surfaces. They are not usually noticeable on textured surfaces.

4.4 Downsampling and Upsampling

In order to construct a Laplacian pyramid of a function defined on a triangulation, the basic operations of downsampling, upsampling, and convolution must be redefined. As mentioned before, the given remeshed triangulation becomes the geometry for the residual lowpass, G_0, of the Gaussian pyramid. Afterwards, recursive subdivision is used to generate the geometry of each subsequent level of the pyramid, G_1 through G_{K-1}. Since, by construction, all vertices of the triangulation are even, the residual lowpass level, G_0, is two-colorable. The butterfly subdivision process neither adds nor removes edges from existing vertices and all new vertices are even, therefore, all subsequent levels of the pyramid are also two-colorable.

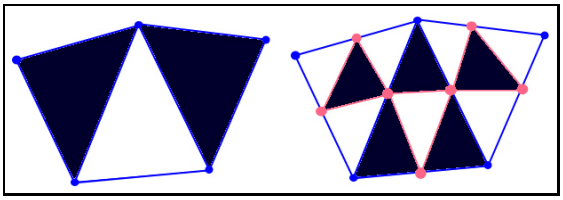

Figure 2: Color inheritance: G_{k-1} (left) and G_k (right).

Each face of the residual lowpass geometry is marked as either *black* or *white* so that no two adjacent faces have the same color. When a face is subdivided by the butterfly scheme, each face is subdivided into four faces. The center face inherits the color of its parent, and the three outer faces inherit the color which is the opposite of its parent's. See Figure 4.4.

Downsampling, populates each cell of G_k with the value of the center child of G_k in G_{k+1}. Upsampling, populates each white cell in G_k with the value of its parent in G_{k-1}, and each black cell in G_k with zeros.

4.5 Gaussian Convolution

In the case of a two-dimensional image, the value of a pixel after discrete convolution with a given kernel can be computed as a weighted sum of the values of the pixel's neighbors, where the kernel defines the weighting function. The computation is straightforward because all of the pixels are the same size and their spacing is uniform. In the case of a surface, in place of pixels we have faces, and convolution must be redefined to take into account both the size and spacing of neighboring triangular faces. In the implementation described in this paper, the area of a face is used as a measure of its size, and the distance between a pair of faces is defined as the straight-line Euclidean distance between the centroid of each face. Given that this distance is only used for convolution with a Gaussian, which has compact support, this definition of distance is a reasonable approximation.

In order to compute the value of face, i, after convolution with a Gaussian kernel, we must define the kernel weights of faces in its neighborhood, $\Omega(i)$. The kernel weight of face j relative to face i for $j \in \Omega(i)$ is:

$$W_{ij} = g\left(\frac{d(i,j)}{\sigma}\right) a_j \tag{5}$$

where $g(x) = e^{-x^2/2}$, $d(i,j)$ is the Euclidean distance between faces i and j, and σ is the maximum Euclidean distance between all pairs of adjacent faces.

In the expression for W_{ij}, dividing by σ dilates g, a Gaussian of unit standard deviation, so that it is (in effect) a Gaussian of standard deviation σ. 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 design choice (compared to the alternative of setting σ equal to the average distance between adjacent faces), which minimizes the aliasing introduced by downsampling, yet remains faithful to Burt's formulation[Burt 1981]. As in the case of a two-dimensional image, we want the sum of the kernel weights to be unity, so we divide the convolution by the factor, $\sum_{j\in\Omega(i)} W_{ij}$. The value of face i after convolution is therefore:

$$v'_i = \frac{\sum_{j\in\Omega(i)} W_{ij} v_j}{\sum_{j\in\Omega(i)} W_{ij}} \tag{6}$$

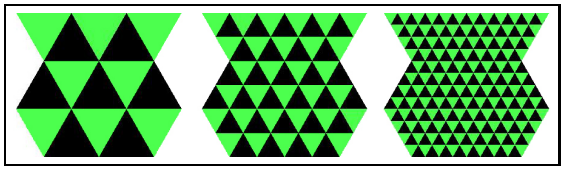

Figure 3: Geometry of G_0, G_1, and G_2 for a 12×12 sample image with $K = 3$.

where v_j is the value of face j before convolution.

We define the neighborhood, $\Omega(i)$, as i, all faces adjacent to i, and all faces adjacent to those faces recursively with the recursion ending in each direction when the distance between the face in question and i, is greater than 3σ. The weights for any faces farther than this distance very quickly approach zero.

4.6 Mapping Rectangular to Triangular Grids

Prior to building the analysis pyramid, we map the sample image from the given rectangular array of pixels to a pair of triangular grids. Each triangular grid is a set of rows (or columns) of congruent, equilateral triangles. The given sample image must be $m \times m$ pixels where $m = r2^{(K-1)}$, K is the desired number of pyramid levels, and r is a positive integer. Then, the sample image would be mapped to a triangular grid with m rows. This triangular grid will be the geometry of the finest resolution level of the pyramid. Therefore, it needs to be the result of $K-1$ applications of the butterfly subdivision.

For example, consider a 12×12 pixel sample image with $K = 3$. Then the geometry of G_2 would have 12 rows of triangles, G_1 would have 12/2 = 6 rows, and G_0 would have 12/4 = 3 rows. See Figure 3.

Because there does not exist a one-to-one mapping between the rectangular grid of pixels and the grid of equilateral triangles, we use bilinear interpolation prior to resampling. In keeping with our assumption of isotropy, we actually interpolate the image onto two triangulations–the second is a 90° rotation of the first. This effectively doubles our sample size. Additionally, when evaluating a neighborhood's similarity, we consider only the best match of three 60° rotations of the neighborhood.

4.7 Reduce and Project

With resampling and convolution defined for the surface, suitable definitions for *reduce* and *project* can be given. Reduce is straightforward: convolve and downsample. Project, however, must apply a normalization factor to ensure the correct range of values in its result. Recall that when a surface is upsampled, not all of the resulting child faces will be nonzero-valued; only half of the faces of the child level inherit the value of their parent. In order to preserve the overall brightness of the input, we compute a normalization factor for each output face:

$$W_i' = \sum_{j \in \Omega(i)} S(j) w_{ij} \tag{7}$$

where $S(j)$ equals unity if the face j inherited its value from its parent. Otherwise, $S(j)$ equals zero.

The value of face i after upsampling and convolution is thus:

$$v_i' = \sum_{j \in \Omega(i)} v_j \left(\frac{w_{ij}}{W_{ij}'} \right). \tag{8}$$

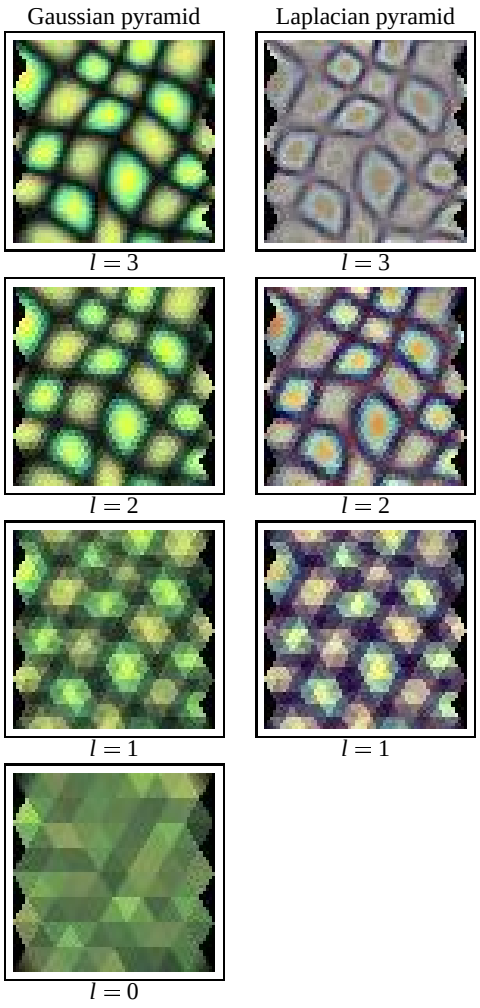

Figure 4: Gaussian and Laplacian pyramids for color texture defined on a triangular grid.

For example Laplacian and Gaussian pyramids computed for a color texture defined on a triangular grid, see Figure 4.

5 Synthesis

After building the analysis and synthesis geometries, and computing the Laplacian pyramid transform of the input texture, $G_A[0]$, the synthesis process can begin. The first step of the synthesis process is to initialize the residual level of the synthesis Gaussian pyramid, $G_S[0]$, with the RGB values of randomly chosen faces of the residual level of the analysis Gaussian pyramid, $G_A[0]$. After this initialization, each face is visited in random order and its value is updated using the non-parametric sampling procedure. When a value is copied from an analysis face to a synthesis face, the RGB values are copied into $G_S[k]$ and the RGB values of the face's four Laplacian children are copied into $L_S[k+1]$. This completes the synthesis of the lowest level of the Gaussian pyramid. Each subsequent Gaussian pyramid level, $G_S[k]$, is initialized by a single step of the inverse Laplacian pyramid transform:

$$G_S[k] \leftarrow L_S[k] + project(G_S[k-1]). \tag{9}$$

This operation, by itself, produces a good initial approximation of the next Gaussian pyramid level. This initial approximation is then updated by random visits to each face, using the non-parametric sampling procedure. The synthesis process is repeated until the final Gaussian level, $G_S[K-1]$ is generated. This level represents the final texture. Pseudo code for a somewhat simplified, one dimensional monochromatic version of the above coarse-to-fine texture synthesis process is shown in Algorithm 1.

SynthesizeTexture$(G_A[0])$
begin
  $\langle L_A, G_A \rangle \leftarrow LaplacianPyramidTransform(G_A[0])$
  $K \leftarrow \log_2(size(G_A[0]))$
  $G_S[0] = G_A[0]$
  for $k = 1$ **to** $K-1$ **do**
    for $update = 1$ **to** $N\,2^k$ **do**
      $i = random(2^k)$
      $\Omega_S = Neighborhood(G_S[k], i)$
      $\langle g, l_0, l_1 \rangle \leftarrow GreedySample(L_A, G_A, k, \Omega_S)$
      $G_S[k][i] \leftarrow g$
      $L_S[k+1][2i] \leftarrow l_0$
      $L_S[k+1][2i+1] \leftarrow l_1$
    end
    $G_S[k+1] \leftarrow project(G_S[k]) + L_S[k+1]$
  end
  return $G_S(K-1)$
end

See Figure 5 for an example of how the texture synthesis process proceeds from coarse-to-fine scales.

5.1 Non-parametric Sampling Procedure

A face's k-distant neighborhood 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. We construct these neighborhoods as ordered sets based on an adjacency relation where a neighborhood about face i is defined to be

$$\{a_1, a_2, a_3, ..., a_p, b_1, b_2, b_3, ..., b_p, c_1, c_2, c_3, ..., c_p\}$$

where a_1, b_1, andc_1 are adjacent to (and in clockwise order around) face i. Additionally, the ordering is such that the adjacency relationship between a_j and a_k is the same as the adjacency relationship between b_j and b_k and between c_j and c_k. See Figure 5.1.

The RGB values of each face in an ordered neighborhood with cardinality $3p$, can be thought of as a point in $9p$ dimensional space. A nearest neighbor query can update face i with neighborhood $\Omega_S(i)$ in the synthesis pyramid by replacing its RGB value with that of face j with neighborhood $\Omega_A(j)$ in the analysis pyramid, such that the $9p$ dimensional Euclidean distance between $\Omega_S(i)$ and $\Omega_A(j)$ is minimum. Unfortunately, a policy of always choosing the face from the analysis pyramid which minimizes the error distance, will not, in general, preserve the first-order statistics of the sample texture. The most obvious way to preserve the first-order statistics is to use sampling without replacement. Unfortunately, this would overconstrain the nearest neighbor query toward the end of each pass through the synthesis pyramid level. We have observed that the use of sampling without replacement produces speckling in the synthesized texture. A good compromise between faithful replication of the first-order statistics and over constraining the nearest neighbor query is to scale each error distance by a factor of $2^{(c[j]-c_{min})}$,

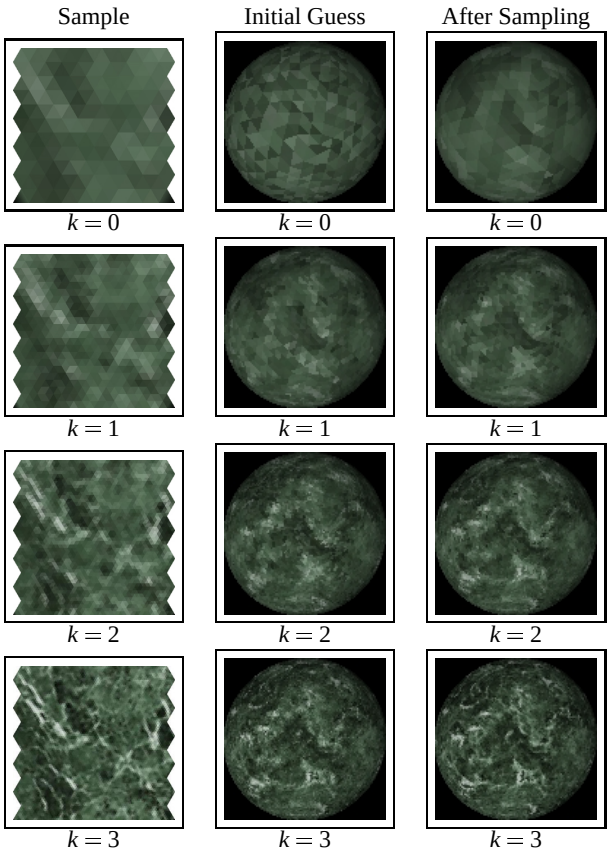

Figure 5: Coarse-to-fine synthesis of green marble texture on sphere using incremental inverse Laplacian pyramid transform to generate initial approximations.

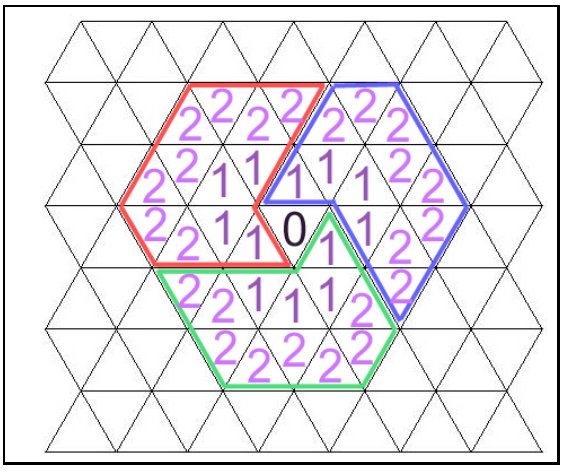

Figure 6: Neighborhood stencil showing three-fold symmetry. There are 12 one-distant neighbors and 24 two-distant neighbors.

where $c[j]$ is the number of times that face j has already been chosen during the update process, and c_{min} is the least number of times that any analysis face has been chosen.

Since we assume isotropic textures, we exploit the three-fold symmetry of the neighborhood, by defining the distance between $\Omega_S(i)$ and $\Omega_A(j)$ as the minimum of the error distances between $\Omega_A(j)$ and three 60° rotations of $\Omega_S(i)$.

On the analysis side, all vertices have a valence of six except for those on the boundary. Analysis neighborhoods that include boundary vertices are not included in the nearest neighbor sampling process. On the synthesis side, the meshes are constructed so that the vast majority of vertices have a valence of six. For simplicity, we assume that all have a valence of six. A vertex with a valence greater than six, by our method of neighborhood construction, causes a gap in the neighborhood. A vertex with a valence less than six causes the neighborhood to overlap with some faces being included more than once. We ignore these errors. Neighborhoods on the synthesis side that contain boundary vertices are treated specially. All "missing" neighbors are considered as "wild cards" which perfectly match any corresponding neighbor in the analysis neighborhood.

Pseudo code for a somewhat simplified, one dimensional monochromatic version of the above sampling procedure is shown in Algorithm 2.

GreedySample(L_A, G_A, k, Ω_S)
begin
  $c'_{min} \leftarrow$ MAXINT
  $d_{min} =$ MAXFLOAT
  for $j = 1$ **to** 2^k **do**
    if $c[j] < c_{min}$ **then**
      $c'_{min} \leftarrow c[j]$
    end
    $\Omega_A \leftarrow Neighborhood(G_A[k], j)$
    $d \leftarrow Distance(\Omega_A, \Omega_S) 2^{(c[j]-c_{min})}$
    if $(d < d_{min})$ **then**
      $j_{opt} \leftarrow j$
      $d_{min} \leftarrow d$
    end
  end
  $c_{min} \leftarrow c'_{min}$
  $c[j_{opt}] \leftarrow c[j_{opt}] + 1$
  return $\langle G_A[k][j_{opt}], L_A[k+1][2j_{opt}], L_A[k+1][2j_{opt}+1] \rangle$
end

---

6 Experimental Results

All of the results shown in this section use four levels of resolution, with a neighborhood size of 12 in the lowest resolution level and 36 in all other levels. Results from other researchers were downloaded from each named researcher's web site.

7 Conclusion

We have described a fully automatic method of synthesizing isotropic textures on subdivision surfaces from sample images. Unlike previous approaches, texture synthesis is accomplished in a coarse-to-fine fashion by constructing both Gaussian and Laplacian pyramid representations of the synthetic texture. The inverse Laplacian pyramid transform is used to generate first approximations to the texture at each level of the associated Gaussian pyramid. These approximations are refined using a modified nearest neighbor search process which preserves the first order statistics of the sample texture.

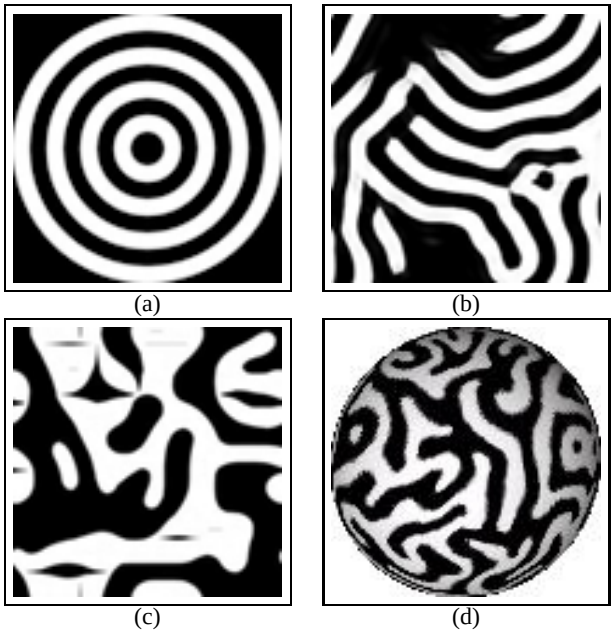

Figure 7: (a) Bullseye pattern from Simoncelli and Portilla. (b) Synthesis result due to Simoncelli and Portilla. (c) Synthesis result due to Wei and Levoy. (d) Our result.

References

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

BURT, P. J. 1981. Fast filter transforms for image processing. *The Visual Computer 16*, 20–51.

DEBONET, J. S. 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH 97*, 361–368.

EBERT, D. S., MUSGRAVE, F., PEACHEY, D., PERLIN, K., AND WORLEY, S. 1994. *Texturing & Modeling: A Procedural Approach*. Academic Press.

EFROS, A., AND FREEMAN, W. 2001. Image quilting for texture synthesis and transfer. In *SIGGRAPH 01*, 341–346.

EFROS, A., AND LEUNG, T. 1999. Texture synthesis by non-parametric sampling. In *Intl. Conf. on Computer Vision*, vol. 2, 1033–1038.

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

PRAUN, E., FINKLESTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *SIGGRAPH 00*, 465–470.

SIMONCELLI, E., AND PORTILLA, J. 1998. Texture characterization via joint statistics of wavelet coefficient magnitudes. In *Fifth Intl. Conf. on Image Proc.*, vol. 1, 62–66.

TURK, G. 1992. Re-tiling polygonal surfaces. In *SIGGRAPH 92*, 55–64.

WEI, L., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH 00*, 479–488.

WEI, L., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH 01*, 355–360.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1996. Interpolating subdivision for meshes with arbitrary topology. In *SIGGRAPH 96*, 189–192.

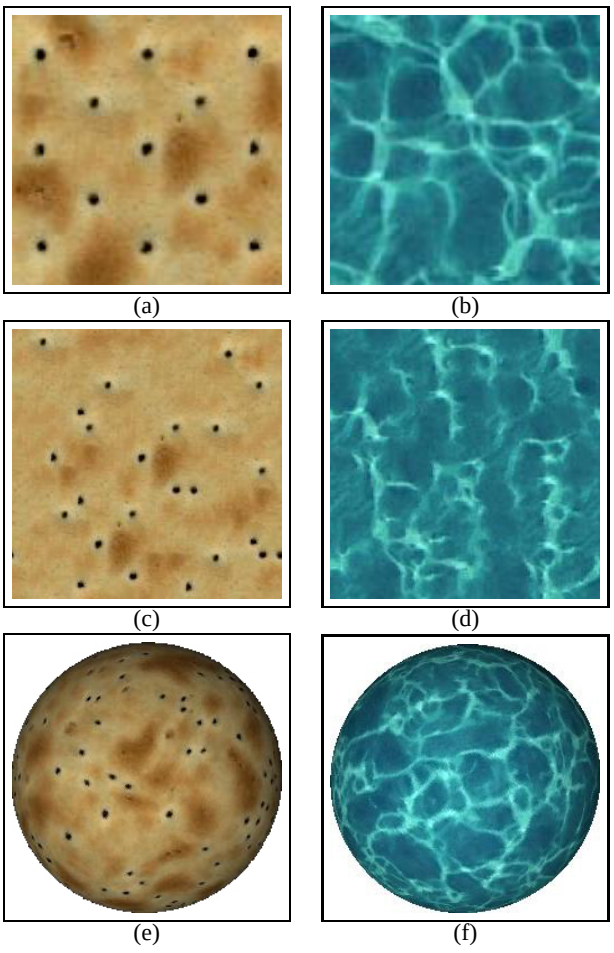

Figure 8: (a) Saltine cracker. (b) Swimming pool bottom. (c) Synthesis result due to Wei and Levoy. (d) Synthesis result due to Wei and Levoy. (e) Our result. (f) Our result.