

Local resampling for patch-based texture synthesis in vector fields

Renjie Chen, Ligang Liu*, Guangchang Dong

Department of Mathematics
Zhejiang University, China
E-mail: ligangliu@zju.edu.cn
*Corresponding author

Abstract: In order to synthesize distortionless texture on surfaces, we develop a direct and accurate approach for local resampling in vector fields, and then use the approach to synthesize textures on 2-D manifold surfaces directly from a texture exemplar. Regular-grid patches produced by the local resampling are used as building blocks for texture synthesis. Then texture optimization and patch-based sampling are generalized to synthesize texture directly in vector fields. The first scheme can create texture of higher quality; however the second scheme is faster and simpler and works well for a wide range of textures. Users can control the vector field on the mesh to generate textures with local variations including the orientation and scale. Many experimental results are presented to demonstrate the ease of use and reliable results provided by our system.

Keywords: Texture synthesis, vector field, local resampling, texture optimization, patch-based sampling

1 INTRODUCTION

Texture synthesis on surfaces has significantly increased the ease of mapping image details on arbitrary meshes over the last decade. There has been lots of work towards synthesizing textures directly on surfaces which naturally reduce distortion inevitably introduced by texture mapping.

However, most previous approaches didn't work well enough for highly-structured and large-scaled texture for complicated surfaces. Our key insight to this problem is that most of them locally flatten a small portion of surfaces, and then extract grid-patches directly from it. These grid-patches, which are counterpart of pixel patches in images, are then used to synthesize current pixels or patches. These approaches may bring severe distortion for large patches, as it simply ignore the flattened vector fields generated at the same time that surfaces are locally flattened.

In this paper, we instead use a more direct and intuitive method for local resampling. Our solution obtains all points of the local grid-patches by tracing integral curves in the vector fields. This approach can produce local grid-patches of less distortion especially for large patches and very complicated surfaces.

Using our local resampling, the patch-based texture synthesis algorithm can be easily generalized to surfaces. In this paper, we first present our extension of texture optimization [10] to 2-D manifold surfaces, and then we also present a non-trivial extension of the patch-based sampling approach proposed in [13, 5]. Thanks to our local

resampling, users can control the vector field on the mesh to generate textures with local variations including orientation and scale. These approaches can also be applied to planar vector fields to produce good results. While texture optimization can produce results of better quality, however, it's much slower than the patch-sampling approach due to its iterative scheme. To further make our method more practical, we develop a clustering-based acceleration method for the situation where PCA for very high dimensional space is impractical. With our general acceleration, patch-sampling approach can achieve near real-time applications.

The rest of the paper is organized as follows. In section 2, we briefly review previous work. In section 3, we explain our local resampling technique in detail. Our optimization-based texture synthesis and patch-based sampling approaches are presented in section 4 and section 5 respectively. Experimental results are presented in section 6. We conclude the paper in section 7 with the summary and future work.

2 PREVIOUS WORK

2.1 Texture synthesis

Example-based texture synthesis has been widely recognized as an important research topic in computer

graphics. Texture synthesis techniques can be broadly categorized into local approaches and global approaches.

Local approaches work by aggressively synthesizing a minimal unit at a time. Based on the minimal synthesizing unit, they can further be classified as pixel-based or patch-based. In pixel-based approaches [6, 14, 2], to color the current pixel, the texture exemplar is searched for a pixel who has similar neighborhood with the currently synthesizing pixel. Patch-based approaches [22, 13, 5, 3, 27] synthesize textures by copying a patch from the texture sample at a time. These approaches are orders of magnitude faster than pixel-based ones and produce results of much better quality.

On the other hand, global methods evolve the entire texture as a whole, based on some criteria for evaluating similarity with respect to the texture exemplar. Texture optimization [10] merges locally defined similarity measure into a global metric which is so called texture energy. Large neighborhoods are adjusted interactively during optimization, so texture energy reduces gradually and the output becomes more and more like the exemplar.

2.2 Texture synthesis on surfaces

There has been a lot of work towards synthesizing textures directly on surfaces. Distortion introduced by texture mapping reduces by synthesizing directly on surfaces.

The pixel-based approach is simultaneously generalized by [21] and [20] to synthesizing textures on 3D surfaces. These two approaches synthesize the texture by coloring individual vertices in a densely resampled mesh. This method has been extended to synthesize bidirectional texture functions (BTF) in [19] and generate progressively-variant textures in [25]. Ref [23] divides the surface into a number of charts and synthesizes texture for each chart by applying an image-based approach. Ref [24] generalizes their image-based jump map texture synthesis algorithm, as it randomly copy pixels from the coherent candidate set without further matching, the results are of poor quality. Although their patch-based extension of jump-map can produce much better result for image, it can hardly be extended to 3D surfaces since jumps would have to take place at vertices which is infeasible.

The patch-based approach is first introduced by lapped texture [17], which is a generalization of the Chaos Mosaic to surfaces [22]. Their method works by randomly pasting user-specified texture patches with irregular shape onto the mesh. Patch boundaries are alpha-blended to hide any seams. Unfortunately since texture patches do not match on their boundaries, this method does not work for more structured textures with sharp discontinuities. Ref [28] further generalizes lapped texture to synthesize anisotropic solid texture. Ref [18] has proposed a multi-scale hierarchical algorithm by stitching small texture patches and matching them on the boundaries. However, the running time is related to the number of resulting patches and synthesis may take a few tens of minutes. A faster though not quite interactive approach is proposed by [15]. The approach computes a set of texon labels for each pixel of the sample by clustering Gaussian-weighted neighborhoods and searching the patches by label matching. However, larger

input samples require more texon labels, which make matching more computationally expensive, and thus slows the running time significantly. A similar approach was developed by [4], decomposing the texture into mostly user-identified patches of texture called texture particles. Ref [16] proposes a hybrid algorithm which initially places large patches and uses a pixel-based synthesis algorithm to reduce boundary artifacts.

Ref [7] has extended Wang tiles [3] to surfaces, a mesh surface is first conformally mapped to a polycube with low distortion, and the surface of the polycube is tiled with Wang tile set, and the surface of polycube is mapped back to the original surface at last. The texture orientation of the final result is exclusively determined by the mapping polycube, so leaving users no control at all. And as most portions on the original surface mapped to single cubes are actually of different acreage, polycube can hardly get rid of mapping distortion.

Texture optimization [10] has recently been extended to 3D surfaces [8, 11]. In ref [8], they replace the original optimization by least square with a discrete solver, and further use K-coherence set [19] to accelerate the M-Step. With these hardware-friendly improvements, their approach achieves real-time, and avoids blending effects in the results. Texture optimize has further been used for synthesizing solid texture from 2D texture exemplar[29], multiscale texture synthesis from multiple texture exemplars[30], and inverse texture synthesis[31].

Among all these algorithms, [21, 20, 19, 25, 17, 11, 8] extract local neighbor grids by flatten surfaces locally and resample from it. As is said in [17], the local flatten scheme has to be done carefully, otherwise may introduce severe distortion even flipped triangles. More complicated local parameterization should be adopted to avoid these situations. Meanwhile, the 3D vector field on the mesh is flattened, and a more reasonable resampling way should be following the flattened vector field. However, since the most pixel-based approaches use small neighbor grids and the local vector fields rarely change much, so the mapping distortion is always tiny and can be neglected.

In order to generalize patch based texture synthesis algorithm and synthesize distortionless texture on 3D surfaces, we must adopt a more accurate resampling method. Inspired by [23], we have developed our novel local resampling method, and then we generalize both texture optimization [10] and patch-based sampling [13, 5] to synthesize textures on 2-D manifold surfaces.

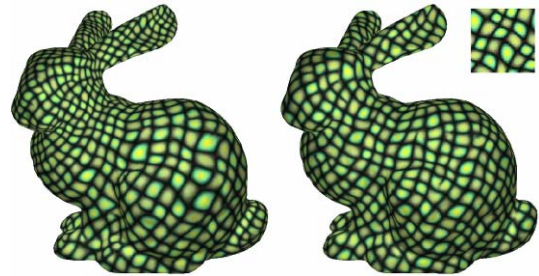


Figure 1. Texture synthesis results on bunny model using our approach. Our approach can generate both uniform texture(right) and non-uniform (progressively-variant) texture(left) according to the specified vector field over the surface.

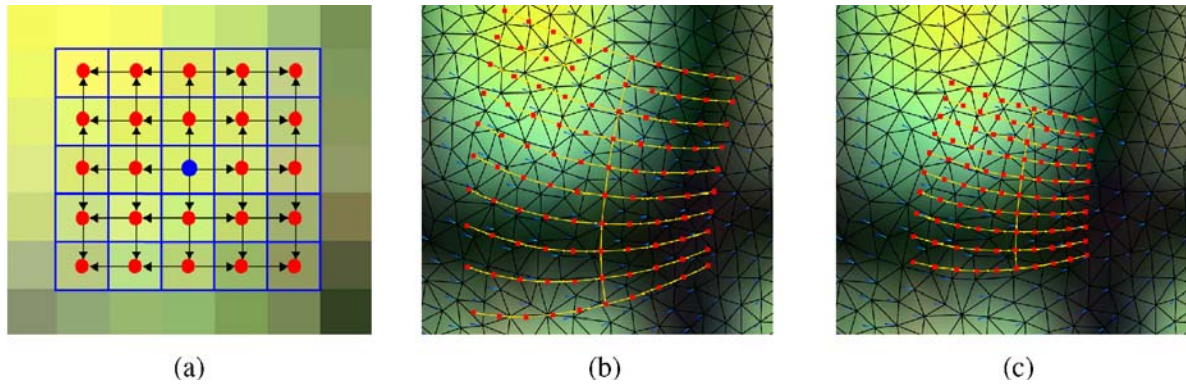


Figure 2. Local resampling. (a) A rectangular neighborhood pattern on texture sample. Each sample location in the neighborhood may be reached by tracing an integral path from its adjacent sample point. (b) Example of uniform local resampling in the vector field. (c) Example of non-uniform local resampling in the same vector field.

3 LOCAL RESAMPLING

Local resampling is necessary for constructing neighborhood used in matching while synthesizing texture. We need to collect a grid of sample points that corresponds to a grid of pixels in the planar patch (Figure 2(a)).

3.1 Vector field generation

Before local resampling, we need to generate a vector field on the surface to specify the local texture orientation. Like [20], we allow a user to specify the direction of the texture at a few vertices and then interpolate the vectors over the other vertices. In our implementation, we use radial basis function approach to interpolate these vectors, in which the approximated geodesic distance [26] is used to compute the distance between two points on the mesh, and then we locally smooth the surface vector field. In addition to the surface vector field orientation (which determines the local texture orientation), we also record its magnitude for each vector, and later, this magnitude will be used to determine the local scale of the texture to be synthesized on the surface. Since our local resampling approach is highly related to the vector field on the surface, and this approach can also be applied to planar vector fields, we refer to surfaces as vector fields. We also refer to the resultant local grid-patches in the vector field as simply resampling patches.

3.2 Resampling in vector field

When synthesizing texture for a small portion of a surface, most of previous work first locally flatten this small portion, and then extract regular-grid patches directly from it. This approach may bring severe distortion for large patches, as it simply ignores the flattened vector field generated at the same time that the surface is locally flattened. In order to solve this problem, we adopt a more direct way that is sampling by tracing integral path in the vector field.

Each smooth vector field deduces its accompanying orthogonal vector field. We adopt a numerical approach to trace a network of orthogonal integral paths at the sample

points locally [9]. For each point to be sampled, we draw two straight paths from the center sample point along the orthogonal integral paths respectively and find the corresponding surface sample points at specified distances. When the path intersects a mesh edge, the line is continued on the adjacent triangle along the same integral path. Once we have obtained a sample point on the integral path, we can compute its neighboring sample point in a similar way, i.e., along the corresponding integral path of current sample point. Specifically, the paths we used here correspond to marching up or down in the texture domain, then left or right to find the sample point (Figure 2(a)). When each of these paths reaches different points on the surface, we use their average points (over the surface) as the final sampling point.

The distance between neighboring sampling points corresponds to the distance between neighboring pixels in the texture example. We can sample the points along the integral paths at a uniform distance. The grid-patches obtained by this resampling method illustrate the orientation variation of the vector field but do not reveal the scale variation of the vector field on the surface, as shown in Figure 2(b). An alternative way is to sample the points along the integral paths non-uniformly. The sampling distance can be determined by the magnitude of the vector at the sampling points. Thus both the orientation variation and scale variation of the vector field can be illustrated in the grid-patch, as shown in Figure 2(c). In this way, we can generate progressively-variant texture on the mesh. In our system, the user can choose either type of texture to synthesize.

When synthesizing each patch in the vector field, we need to first extract its neighborhood from the vector field, and search the texture exemplar for a most similar one, and then the corresponding patch is pasted to the vector field. As shown in Figure 3, when extracting neighbor patches, we obtain the color of each sample point P through the following interpolation:

$$C_p = w_1 C_{v_1} + w_2 C_{v_2} + w_3 C_{v_3}$$

Where C_X is the texture color of point X, and (w_1, w_2, w_3) is the barycentric coordinates of P in the triangle $\Delta P_1 P_2 P_3$ that contains P.

On the other hand, we need to assign color to the vertices of the mesh when a texture patch is obtained and pasted onto it. As shown in Figure 3(c), let V be in the quadrangle consisted of grid points P_1, P_2, P_3, P_4 that contains V. We have $C_V = v_1 C_{P_1} + v_3 C_{P_3} + v_4 C_{P_4}$ and $C_V = u_1 C_{P_1} + u_2 C_{P_2} + u_3 C_{P_3}$ where (u_1, u_2, u_3) and (v_1, v_3, v_4) are the barycentric coordinates of V in the triangle $\Delta P_1 P_2 P_3$ and $\Delta P_1 P_3 P_4$ respectively. Then the color of the vertex on the mesh can be calculated as follows

$$C_V = \frac{(u_1 + v_1)}{2} C_{P_1} + u_2 C_{P_2} + \frac{(u_3 + v_3)}{2} C_{P_3} + v_4 C_{P_4}$$

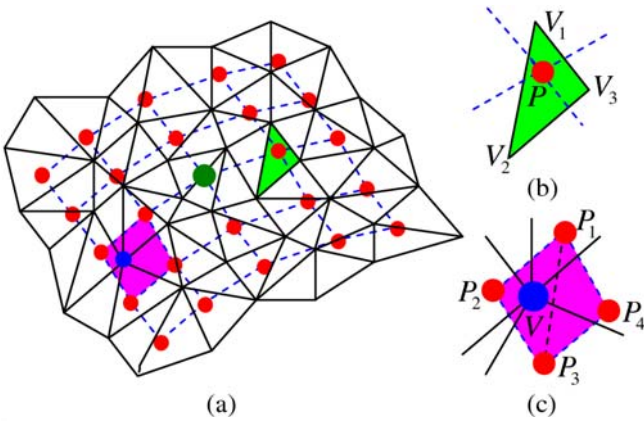


Figure 3. Patch sampling around the green vertex is shown in (a) where the resampling points are shown in red points. The computation of the grid point P in the green triangle $\Delta V_1 V_2 V_3$ is shown in (b). The computation of the vertex V in the pink quadrangle $P_1 P_2 P_3 P_4$ is shown in (c).

4 TEXTURE SYNTHESIS BY OPTIMIZATION

We separate texture generation into two phases: preprocessing and texture synthesis. While preprocessing is relatively slow, it only has to be done once per exemplar and per mesh. At the same time, preprocessing makes the actual texture synthesis process fast.

4.1 Texture preprocessing

The goal of texture preprocessing is to identify the sets of patches in the exemplar that have similar appearance.

4.1.1 Appearance vector

The traditional approach in texture synthesis is to compare color neighborhoods with those of an exemplar [6, 14, 5]. Distance is typically measured by summing squared color differences to compare a synthesized neighborhood and some exemplar neighborhood. As is well known that, L2

distance in RGB-space is a poor metric for measuring similarity between two image patches. So we first transform the exemplar to appearance-space [12], and use L2 distance in appearance-space instead.

As shown in Figure 4, we apply neighborhood projection [12] to the exemplar itself that is replacing each pixel's RGB color with the concatenation of all its neighbors' RGB colors. As we can see, this transformed exemplar is of very high dimension, so we further project this exemplar using PCA to obtain a low-dimensional transformed exemplar. We further incorporate a feature distance map of the exemplar into the appearance-space. After that, each pixel's RGB-color is replaced with appearance-space 'color', which is so called appearance vector. Refer to [12] for more detailed introduction of appearance-space.

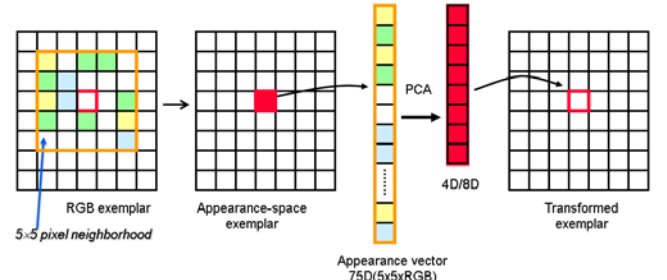


Figure 4. Appearance-space is constructed in the same way as [12].

4.1.2 Analysis of texture patches

During texture synthesis, we need to search the set of all patches with the same size, say $w_B \times w_B$, in the texture exemplar T_l for those patches which are most similar to the current patch B_o , and then one of the most similar patches is used for synthesizing.

Actually we don't need to find the exact n patches which are most similar to the current patch; we are willing to accept n approximate nearest neighbors. This search can be considered as a n approximate nearest neighbors search problem in the high-dimensional space, which consists of texture patches of the same shape and size as B_o . As is revealed by Table 1, we suffer severe high dimension searching problem for the large patch neighborhood we used, and appearance-space technique make this situation even worse. For example, if the dimension of appearance vector is 5, the size of the patch is 32×32 , then the dimension of the searching space would be $5 \times 32 \times 32 = 5120$. Therefore, we reduce the searching dimension with PCA method, however the PCA preprocess perform badly for such high dimension, so we apply a special dimension-reduction method before PCA preprocess. Since each appearance vector actually describes the property of a small neighborhood, we choose some representation pixels and use them to measure the similarity between two texture patches instead of all pixels in the neighborhoods. After dimension-reduction, efficient searching algorithms [1] (i.e., ANN) can be used.

Inspired by the hierarchical clustering accelerating method proposed in [10], we also developed a clustering-based method for direct approximate nearest search in high-dimensional space. We perform k-means clustering on all the texture patches in the exemplar. The centers of all clusters are taken as the candidate set in the search space. Patches, that end up in the same cluster have similar appearance. A large number of clusters reduce the average size of each cluster. During synthesizing, we first search the optimal cluster center within all cluster centers, and then we search for the most similar vectors within that cluster.

Appearance transformation, dimension-reduction or clustering are only done once per texture during the preprocessing step, and the results can be saved for later use.

4.2 Mesh preprocessing

In our approach, we synthesize textures as vertex colors directly over the target mesh surface. Both of our texture synthesis methods need first obtain all regular-grid patches in the vector fields. The goal of mesh preprocessing is to compute the sampling points for all the grid patches in the vector field.

We first randomly and uniformly select some vertices from all of the vertices on the mesh, and these vertices are saved in a set X^\dagger . Each vertex in X^\dagger is then used as grid-patch center from which we calculate all the grid-patches with our local resampling method. When progressively-variant texture is being synthesized, we select these vertices according to the length of local vectors. More vertices are selected where the local vectors are short. The distance between neighboring patch centers p are determined by the size of grid-patch. Like [10], we choose former as $1/4$ latter, so each vertex is covered by 4 patches averagely. All grid-patches are then saved in the resampling patch set Φ . This is done before the synthesis process, in order to make the texture synthesis fast.

4.3 Texture optimization

Let X be the vectorized version of the target texture X that we want to synthesize, which is formed by concatenating the color value of all pixels in X . Let X_p be a sub-vector of X that corresponds to the neighborhood of the pixel P . Further let Z_p be the vectorized pixel neighborhood in the input sample Z , whose appearance is most similar to X_p . Then we define our texture energy in a similar way to [10]:

$$E_t(X; \{Z_p\}) = \sum_{p \in X^\dagger} \omega_p |X_p - Z_p|^2$$

However our texture energy is defined in appearance-space instead of RGB-space. The subset X^\dagger is obtained from the previous mesh preprocessing step. The coefficient ω_p is used for accelerating the subsequent EM iterative operation, and we set it as suggested in [10]:

$$\omega_p = |X_p - Z_p|^{-0.8}$$

Like [10], we minimize the texture energy E_t with EM-like (Expectation-Maximization) algorithm. We first initiate X with random colors. Then we iterate the EM-algorithm until the texture energy stop changing. In M-Step, we minimize $E_t(X; \{Z_p\})$ w.r.t. the set of $\{Z_p\}$. We extract each texture patch from the vector field with corresponding resampling patch, search for the most similar patch in the exemplar, and record both its position and the similar degree $|X_p - Z_p|^2$ which is later used for computing the weight ω_p . In E-Step, we minimize $E_t(X; \{Z_p\})$ w.r.t. X . In our implementation of least square solver, we record both color and weight at each vertex of the mesh. At the beginning of E-Step, both color and weight are cleared, and then texture patches from specified location of the exemplar are pasted to the vector field with weight one by one, at the end the color of each vertex is calculated. When finished synthesizing in appearance-space, we transform the vector field back to RGB-space. Since we have recorded the locations and weights of all patches in the exemplar, we need only paste the texture patches of RGB-space to the vector field, and then textures of high quality are obtained. Algorithm 1 describes the pseudo code for texture optimization algorithm

Algorithm 1. Texture Optimization

```

 $Z_p^0 \leftarrow$  random neighborhood in  $Z \ \forall p \in X^\dagger$ 

for iteration  $n = 0 : N$  do
     $X^{n+1} \leftarrow \arg \min_x E_t(X; \{Z_p^n\})$  // E-Step
     $Z_p^{n+1} \leftarrow$  nearest neighbor of  $X_p^{n+1}$  in  $Z \ \forall p \in X^\dagger$  // M-Step
    if  $Z_p^{n+1} = Z_p^n \ \forall p \in X^\dagger$  then
         $X \leftarrow X^{n+1}$ 
        break
    endif
endfor

```

4.4 Multi-level synthesis

In order to produce higher-quality textures, we adopt a multi-resolution and multi-scale fashion similar to [10, 8]. However, instead of building mesh pyramid, we make use of the Gaussian pyramid of the resampling patches, so we need only a final resultant mesh.

When synthesizing in coarse level, we scale both the resampling patches and texture patches accordingly. Put it concretely, in E-step, after we get a coarse texture patch from the down-sampled exemplar, we up-sample and paste it into the vector field with corresponding resampling patch.

In M-step, we extract texture patch from the vector field according to the down-sampled resampling patch, and search in the corresponding coarse exemplar for match.

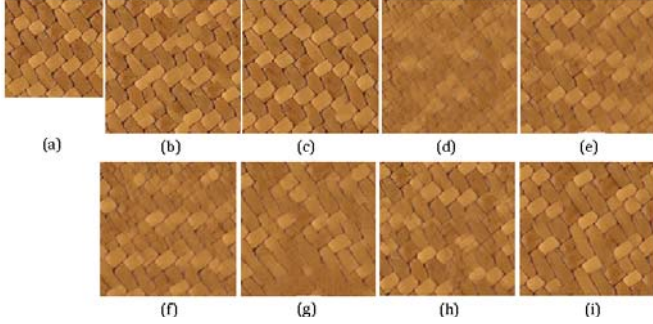


Figure 5. Multi-level synthesis. (a)Input texture. (b)Synthesis result without multi-level. (c) Synthesis result with 3 resolutions and 3 scales.(d)-(i). Different stages to synthesis (c).

For the multi-scale fashion, we proceed local resampling in the vector field with different neighborhood size, and save the resultant resampling patches during mesh preprocessing. The texture in figure 5(c) is generated with 3 resolutions and 3 neighborhood sizes. Figure 5(d) shows synthesis at (1/4 resolution, 8x8 neighborhood). Figure 5(e): (1/2, 16×16). Figure 5(f): (1/2, 8×8). Figure 5(g): (1/1, 32×32). Figure 5(h): (1/1, 16×16). Figure 5(i): (1/1, 8×8). As we can see, multi-level approach obviously produces much better result (in comparison with figure 5(b)).

5 TEXTURE SYNTHESIS BY PATCH-BASED SAMPLING

As our texture optimization is quite slow due to its intrinsic iterative operation, so we present an easier and faster synthesis method by patch-sampling. A key idea of this algorithm is a sampling scheme that uses texture patches with specified sizes of the sample texture as building blocks for texture synthesis. This method produces results of quite good quality, although a little worse than previous optimization approaches.

Our extension of texture synthesis approach based on patch-based sampling has the following advantages:

- **Fast:** Our approach is much faster in run time than both the pixel-based texture synthesis approaches and the other patch-based approaches such as [18] and [15].
- **Parallelizable:** Our approach synthesizes texture on surface by pasting patches in a random order and thus achieves parallelism.

Inspired by the work of [13, 5], our patch-based sampling approach uses rectangular texture patches of the input sample texture T_i as the building blocks for constructing the synthesized mesh M_o , as shown in Figure 6 (a). In each step, we paste a patch B_l of T_i onto M_o (Figure 6(c)). Each patch B_l has a boundary zone E_B of

width W_E , as shown in Figure 6(b). To avoid mismatching features across patch boundaries, we carefully select B_l based on the patches already pasted on M_o . For simplicity, we only use square patches of a prescribed size $w_B \times w_B$.

5.1 Preprocessing

Our patch-sampling approach is also separated into two phases: preprocessing and texture synthesis, and the preprocessing is proceeded in a similar way to previous optimize-based approach. In texture preprocessing, we build search space with the boundary of texture patches instead of

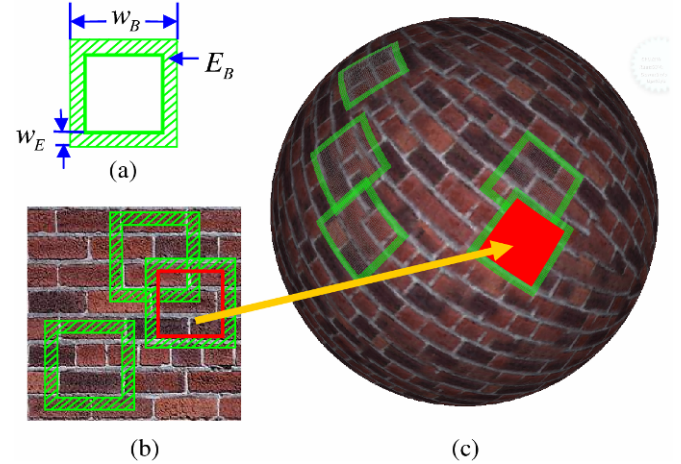


Figure 6. Patch-based sampling strategy. (a) The $w_B \times w_B$ patch B_l has a boundary zone E_B of width W_E . (b) The input texture sample. (c) The synthesized texture on surface. In the input texture sample shown in (b), three patches have boundary zones matching the red texture patch B_l shown in (c) and the red patch is selected.

the whole patch. That's because in patch-based sampling, we need to make neighboring patches match well at the boundary they have in common, and the inner zones of the texture patches don't affect the synthesis quality. In mesh preprocessing, we choose distance of neighboring patch centers as $\frac{1}{2}$ patch size. In this way, each point in the vector field is covered by 2 resampling patches averagely, but the total number of resampling patches is much smaller than optimization approach, so make this synthesizing approach much faster.

5.2 Patch-based sampling

Our patch-based sampling texture synthesis approach proceeds as follows.

Step 1. For each vertex on the mesh, randomly choose a pixel from the exemplar and assign its color to the vertex.

Step 2. Randomly select an unsettled patch B_o from the resampling patch set Φ

Step 3. Select a patch B_l from T_i such that its boundary is the closest to that of B_o .

Step 4. Paste the patch B_l in the region of B_o on the mesh.

Step 5. Repeat steps 2, 3 and 4 until all the vertices of M_o are colored.

Step 6. Perform blending in the boundary zones on M_o .

The synthesis process starts by randomly coloring the vector field. At each step, we try to select the patch from the texture exemplar that best matches along its boundary zone with all textured neighbors in the mesh. We continue until the whole surface is covered.

In [13, 5], texture are synthesized in scanline order, for each patch to be synthesized there are always one or two neighboring patches synthesized earlier, then one can easily select a patch from the exemplar and make it match well with its neighboring patches in their common boundary. However, our approach succeeds by adequately estimation of the untextured boundary of current patch. To make this estimation more accurate we adopt multi-resolution fashion. The coarsest level vector field is initiated by randomly copying pixels from the exemplar, and then patch-based sampling is proceeded. For higher resolution synthesis, we first use the lower resolution information to extrapolate the current resolution level, and then patch-based sampling is performed. In our implementation, we use 3 level hierarchies to synthesis the texture. 3 level hierarchy works well for most of the examples in our experiments.

Our approach synthesizes the texture over the resampling patch set in random order. Note that we can also synthesize the texture over the patch set in a breadth-first order, which is similar to [13, 5], in each step we try to find a patch synthesized earlier and synthesize its untextured neighboring patches. In our experiments, the synthesis results using random order are as good as the ones using breadth-first order. For a planar rectangular region with a horizontal constant vector field, we compare the synthesis results between the approach of [13] and our approach. Figure 7 shows an example comparing the results.

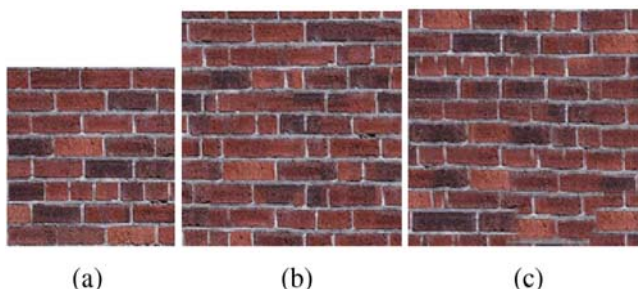


Figure 7. Texture synthesis results comparison. (a) Input texture. (b) Synthesis result using the approach of [13]. (c) Synthesis result using our approach with random order. For most textures, we have found that random-order works well.

6 EXPERIMENTAL RESULTS

We have tested both of our texture synthesis algorithms on a variety of textures and models. Parameters are important for

synthesizing texture. The size of the texture patch affects how well the synthesized texture captures the local characteristics of the texture exemplar. For texture optimization, we use three resolution levels and successive neighborhood sizes of 32×32 , 16×16 , and 8×8 pixels for most texture, and for larger structured texture, we use 40×40 for the largest neighborhood size, and other resolutions and neighborhood level are scaled accordingly. For patch-based sampling, we use similar parameters and the width of patch boundary w_E is usually set to 4.

In figure 8, we show several examples for synthesizing different textures over different planar vector fields, and compare them with the results from some other techniques. Note that different from [10], our method can produce both uniform and progressive-variant textures for the same vector fields, and users can further specify how the scale of texture change over the vector fields. Our results look quite good and preserve the orientation of the vector fields, and scale variation is revealed. Compare (c) and (e) with (d) and (f) respectively, we can see that our technique produces crispier image quality, which we attribute to the local resampling approach.

Figure 1 shows the results of uniform and non-uniform texture synthesis over the bunny mesh model respectively. Note that the progressively-variant texture (Figure 1(a)) has the same orientation with the uniform texture result (Figure 1(b)), but differs in the scale variation. Figure 9 compares our synthesis result with that from ref[12]. It's hard to reproduce the same vector field, so the texture orientation in (b) is slightly different from (a), but we can tell that our method produce better quality by comparing the results near the tail and left ear of the bunny model.

Different textures are synthesized on the head model shown in Figure 10. Several examples for different combinations of meshes and textures are shown in Figure 11. The first 5 results in Figure 11 and the first 2 results in Figure 10 are generated with texture optimization, while the rest are generated with patch-based sampling.

The texture optimization approach produces results of state-of-the-art. Patch-based sampling also produces results of quite good quality for many texture, although it may still produce artifacts on the edges of patches for some texture. However the latter is much faster.

Table 1 summarizes the performance of both algorithms with all kinds of acceleration techniques applied. As can be seen in the table, the method using PCA+ANN in our approach performs better than the other methods.

Analysis/Synthesis(s)	Exhaustive	PCA+ANN	PCA+Clustering	Clustering
Optimization	0.55/1521.40	3.83/24.81	4.55/31.63	7.88/55.57
Patch Sampling	0.55/76.07	9.64/1.80	9.73/3.06	8.17/5.25

Table 1. Timing comparison between different acceleration methods. In the test, we use texture sample of size 96×96 , and patch size is 32×32 , the dimension of appearance vector is 5 including feature-distance map, and the mesh has 196k vertices, edge size is 4 for patch-based sampling, and for optimization, we iterate the EM-algorithm for 10 times. The resultant dimension of PCA is 30. All performance timings are measured on Pentium 4 3.0GHz with 1G RAM.

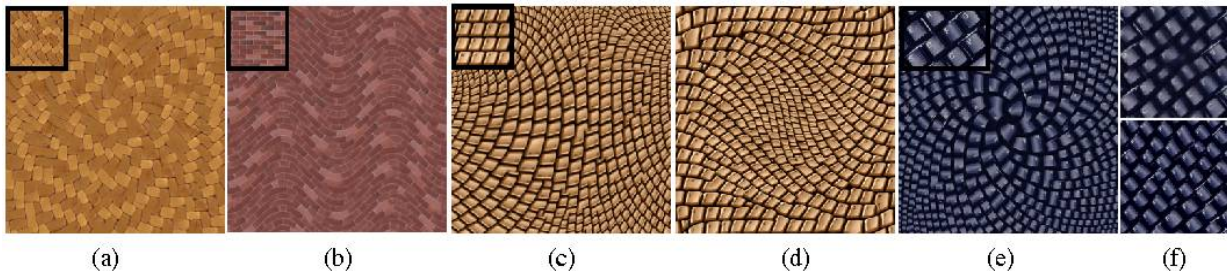


Figure 8. Texture synthesis results over different planar vector fields and comparison with various other techniques. Results for other techniques were obtained from their web pages. (a) Uniform texture over radial vector field; (b) Uniform texture over sin-shaped vector field; (c) Non-uniform texture over user-specified vector field; (d) Results from [10]; (e) Non-uniform texture over the ring-shaped vector field; (f) Results from [10](upper) and [12].

7 CONCLUSIONS

In order to synthesize distortionless texture on surfaces, we present an intuitive and intrinsic local resampling algorithm in vector fields, and extend both texture optimization and patch-based sampling texture synthesis algorithms to 2-D manifold surfaces. In comparison with the local flatten method, our local resampling approach has much less distortion. Thanks to the local resampling, our new texture synthesis method produces results of higher quality than previous work, especially when synthesizing highly-structured and large-scaled texture on complicated surfaces. With our system, users are free to control the texture orientation by specifying different kinds of vector fields on the surface.

There are a few drawbacks in our approach. As mentioned in [8,10], least square solver will introduce blurry blending problem for texture optimization. In the other hand, although patch-based sampling is much faster, it needs further acceleration to satisfy real-time application, and patch seams become noticeable when viewing highly structured textures closely. One of our future work is further acceleration of our texture synthesis methods, especially the texture optimization approach. Note that both of our texture synthesis methods are proceeded in random order, so it shouldn't be too complicated to implement our methods on modern hardware.

ACKNOWLEDGEMENT

This work is supported by National Natural Science Foundation of China (No. 60776799).

REFERENCES

- [1] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Andwu, 1998. An optimal algorithm for approximate nearest neighbor searching. *Journal of ACM*, 45:891–923.
- [2] M. Ashikhmin, 2001. Synthesizing natural textures. In *Proc. of Symposium on Interactive 3D graphics*, pages 217–226.
- [3] M. Cohen, J. Shade, S. Hiller, and O. Deussen, 2003. Wang tiles for image and texture generation. In *Proc. of SIGGRAPH*, pages 287–294.
- [4] J. Dischler, K. Maritaud, B. Levy, and D. Ghazanfarpour, 2002. Texture particles. In *Proc. of Eurographics*, pages 401–410.
- [5] A. Efros and W. Freeman, 2001. Image quilting for texture synthesis and transfer. In *Proc. of SIGGRAPH*, pages 341–346.
- [6] A. Efros and T. Leung, 1999. Texture synthesis by non-parametric sampling. *International Conference on Computer Vision*, 2(9):1033–1038.
- [7] C.-W. Fu and M.-K. Leung, 2005. Texture tiling on arbitrary topological surfaces. In *Proc. of Eurographics Symposium on Rendering*, pages 99–104.
- [8] J. Han, K. Zhou, L.-Y. Wei, M. Gong, H. Bao, X. Zhang, and B. Guo, 2006. Fast example-based surface texture synthesis via discrete optimization. In *Visual Computer*, pages 918–925.
- [9] B. Jobard and W. Lefebvre, 1997. Creating evenly-spaced streamlines of arbitrary density. In *Proc. of Eurographics Workshop on Visualization in Scientific Computing*, pages 45–55.
- [10] V. Kwatra, I. Essa, A. Bobick, and A. Kwatra, 2005. Texture optimization for example-based synthesis. In *Proc. of SIGGRAPH*, pages 795–802.
- [11] V. Kwatra, D. Adalsteinsson, T. Kim, N. Kwatra, M. Carlson and M. Lin, 2007. Texture Fluids. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):939–952.
- [12] S. Lefebvre and H. Hoppe, 2006. Appearance-space texture synthesis. In *Proc. of SIGGRAPH*.
- [13] L. Liang, C. Liu, Y. Xu, B. Guo, and H. Shum, 2001. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3):127–150.
- [14] L. Wei and M. Levoy, 2000. Fast texture synthesis using tree-structured vector quantization. In *Proc. of SIGGRAPH*, pages 479–488.
- [15] S. Magda and D. Kriegman, 2003. Fast texture synthesis on arbitrary meshes. In *Proc. of Eurographics Symposium on Rendering*, pages 82–89.
- [16] A. Nealen and M. Alexa, 2003. Hybrid texture synthesis. In *Proc. of Eurographics Symposium on Rendering*, pages 97–105.
- [17] E. Praun, A. Finkelstein, and H. Hoppe, 2000. Lapped textures. In *Proc. of SIGGRAPH*, pages 465–470.
- [18] C. Soler, M. Cani, and A. Angelidis, 2002. Hierarchical pattern mapping. In *Proc. of SIGGRAPH*, pages 673–680.
- [19] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H. Shum, 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. In *Proc. of SIGGRAPH*, pages 665–672.
- [20] G. Turk, 2001. Texture synthesis on surfaces. In *Proc. of SIGGRAPH*, pages 347–354.
- [21] L. Wei and M. Levoy, 2001. Texture synthesis over arbitrary manifold surfaces. In *Proc. of SIGGRAPH*, pages 355–360.

- [22] Y. Xu, B. Guo, and H. Shum, 2000. Chaos mosaic: fast and memory efficient texture synthesis. In Technical Report MSR-TR-2000-32 of Microsoft Research.
- [23] L. Ying, A. Hertzmann, H. Biermann, and D. Zorin, 2001. Texture and shape synthesis on surfaces. In Proc. of Eurographics Symposium on Rendering, pages 301–312.
- [24] S. Zelinka and M. Garland, 2003. Interactive texture synthesis on surfaces using jump maps. In Proc. of Eurographics Symposium on Rendering, pages 90–96.
- [25] J. Zhang, K. Zhou, L. Velho, B. Guo, and H.-Y. Shum, 2003. Synthesis of progressively variant texture on arbitrary surfaces. In Proc. of SIGGRAPH, pages 295–302.
- [26] G. Zigelman, R. Kimmel, and N. Kiryati, 2002. Texture mapping using surface flattening via multidimensional scaling. *IEEE Transactions on Visualization and Computer Graphics*, 8(2):198–207.
- [27] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick, 2003. Graphcut textures: Image and video synthesis using graph cuts. In Proc. of SIGGRAPH, pages 277–286.
- [28] K. Takayama, M. Okabe, T. Ijiri, T. Igarashi, 2008. Lapped solid textures: filling a model with anisotropic textures. In Proc. of SIGGRAPH, pages 1-9.
- [29] J. Kopf, C. Fu, D. Cohen-or, O. Deussen, D. Lischinski, T. Wong, 2007. Solid texture synthesis from 2D exemplars. In Proc. of SIGGRAPH, pages 2.
- [30] C. Han, E. Risser, R. Ramamoorthi, E. Grinspun, 2008. Multiscale texture synthesis. In Proc. of SIGGRAPH, pages 1-8.
- [31] L. Wei, J. Han, K. Zhou, H. Bao, B. Guo, H. Shum, 2008. Inverse texture synthesis. In Proc. of SIGGRAPH, pages 1-9.

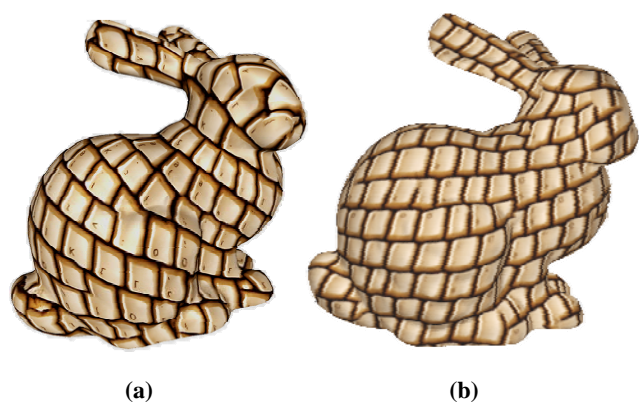


Figure 9. Comparison of texture synthesis on the bunny model. (a) Result of [12]; (b) our result.

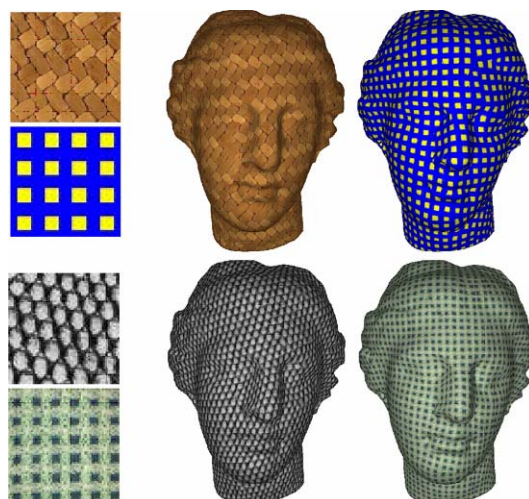


Figure 10. Synthesizing different textures on a head



Figure 11. Some texture synthesis results for different meshes and textures.