Local resampling for patch-based texture synthesis in vector fields

Renjie Chen, Ligang Liu* and Guangchang Dong

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

Abstract: We develop a direct and accurate approach for local resampling in vector fields, and then use the approach to synthesise textures on 2D 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 optimisation and patch-based sampling are generalised to synthesise texture directly in vector fields. 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 optimisation; patch-based sampling.

Reference to this paper should be made as follows: Chen, R., Liu, L. and Dong, G. (2010) 'Local resampling for patch-based texture synthesis in vector fields', *Int. J. Computer Applications in Technology*, Vol. 38, Nos. 1/2/3, pp.124–133.

Biographical notes: Renjie Chen received his BSc Degree in Applied Mathematics from Zhejiang University, China in June 2005. Now he is a PhD Student at the Department of Mathematics in Zhejiang University, under the cosupervision of Professor Guangchang Dong and Professor Ligang Liu. His research interests include digital geometry modelling and processing, computer graphics and image processing.

Ligang Liu received the BS in Mathematics and PhD in Mathematics from Zhejiang University, China, in 1996 and 2001, respectively. From 2001 to 2004, he was an Associate Researcher at the Internet Graphics Group, Microsoft Research Asia. Since 2004, he has been an Associate Professor in the Department of Mathematics at Zhejiang University. His current research interests include geometric modelling and processing, interactive computer graphics and image processing.

Guangchang Dong received the BS in Mathematics from Zhejiang University, China, in 1950. He has been a Professor at the Department of Applied Mathematics in Zhejiang University, China, since 1978. He has been in the editorial board of *Journal of Partial Differential Equations*, *Chinese Annals of Mathematics*, and so on. He is the Chief Editor of *Applied Mathematics A Journal of Chinese Universities*. His research interests include partial differential equations, approximation theory, geometric modelling, curves and surface.

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 synthesising textures directly on surfaces, which naturally reduce distortion inevitably introduced by texture mapping.

However, most previous approaches did not 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 synthesise current pixels or patches. These approaches may bring severe distortion for large patches, as it simply ignores 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 generalised to surfaces. In this paper, we first present our extension of texture optimisation (Kwatra et al., 2005) to 2D manifold surfaces, and then we also present a non-trivial extension of the patch-based sampling approach proposed in Liang et al. (2001) and Efros and Freeman (2001). 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. Whereas texture optimisation can produce results of better quality, however, it is much slower than the patch-sampling approach owing 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 organised as follows. In Section 2, we briefly review previous work. In Section 3, we explain our local resampling technique in detail. Our optimisation-based texture synthesis and patch-based sampling approaches are presented in Sections 4 and 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 recognised as an important research topic in computer graphics. Texture synthesis techniques can be broadly categorised into local approaches and global approaches.

Local approaches work by aggressively synthesising a minimal unit at a time. Based on the basis of the minimal synthesising unit, they can further be classified as pixel-based or patch-based. In pixel-based approaches (Efros and Leung, 1999; Wei and Levoy, 2000; Ashikhmin, 2001), to colour the current pixel, the texture exemplar is searched for a pixel that has similar neighbourhood with the currently synthesising pixel. Patch-based approaches (Xu et al., 2000; Liang et al., 2001; Efros and Freeman, 2001; Cohen et al., 2003; Kwatra et al., 2003) synthesise 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 synthesise the entire texture as a whole, based on some criteria for evaluating similarity with respect to the texture exemplar. Texture optimisation (Kwatra et al., 2005) merges locally defined similarity measure into a global metric, which is the so-called texture energy. Large neighbourhoods are adjusted interactively during optimisation, 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 synthesising textures directly on surfaces. Distortion introduced by texture mapping reduces by synthesising directly on surfaces.

The pixel-based approach is simultaneously generalised by Wei and Levoy (2001) and Turk (2001) to synthesising textures on 3D surfaces. These two approaches synthesise the texture by colouring individual vertices in a densely resampled mesh. This method has been extended to synthesise Bidirectional Texture Functions (BTFs) in Tong et al. (2002) and generates progressively variant textures in Zhang et al. (2003). Ying et al. (2001) divide the surface into a number of charts and synthesise texture for each chart by applying an image-based approach. Zelinka and Garland (2003) generalise their image-based jump map texture synthesis algorithm, as it randomly copies 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 (Praun et al., 2000), which is a generalisation of the Chaos Mosaic to surfaces (Xu et al., 2000). This 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. Takayama et al. (2008) further generalise lapped texture to synthesise anisotropic solid texture. Soler et al. (2002) have proposed a multiscale 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 Magda and Kriegman (2003). The approach computes a set of texton labels for each pixel of the sample by clustering Gaussian-weighted neighbourhoods and searching the patches by label matching. However, larger input samples require more texton labels, which make matching more computationally expensive, and thus slows the running time significantly. A similar approach was developed by Dischler et al. (2002), decomposing the texture into mostly user-identified patches of texture called texture particles. Nealen and Alexa (2003) propose a hybrid algorithm, which initially places large patches and uses a pixel-based synthesis algorithm to reduce boundary artefacts.

Fu and Leung (2005) have extended Wang tiles (Cohen et al., 2003) 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 optimisation (Kwatra et al., 2005) has recently been extended to 3D surfaces (Han et al., 2006; Kwatra et al., 2007). In Han et al. (2006), they replace the original optimisation by least square with a discrete solver, and further use *K*-coherence set (Tong et al., 2002) to accelerate the M-Step. With these hardware-friendly improvements, their approach achieves real time, and avoids blending effects in the results. Texture optimisation has further been used for synthesising solid texture from 2D texture exemplar (Kopf et al., 2007), multiscale texture synthesis from multiple texture exemplars (Han et al., 2008) and inverse texture synthesis (Wei et al., 2008).

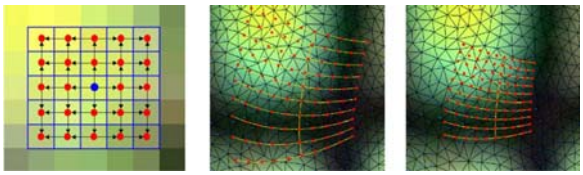
All these algorithms (Wei and Levoy, 2001; Turk, 2001; Tong et al., 2002; Zhang et al., 2003; Praun et al., 2000; Kwatra et al., 2007; Han et al., 2006), extract local neighbour grids by flatten surfaces locally and resample from it. As is said in Praun et al. (2000), the local flatten scheme has to be done carefully, otherwise may introduce severe distortion even flipped triangles. More complicated local parameterisation should be adopted to avoid these situations. Meanwhile, the 3D vector field on the mesh is flattened, and a more reasonable resampling way should follow the flattened vector field. However, since the most pixel-based approaches use small neighbour grids and the local vector fields rarely change much, so the mapping distortion is always tiny and can be neglected.

To generalise patch-based texture synthesis algorithm and synthesise distortionless texture on 3D surfaces, we must adopt a more accurate resampling method. Inspired by Ying et al. (2001), we have developed our novel local resampling method, and then we generalise both texture optimisation (Kwatra et al., 2005) and patch-based sampling (Liang et al., 2001; Efros and Freeman, 2001) to synthesise textures on 2D manifold surfaces.

3 Local resampling

Local resampling is necessary for constructing neighbourhood which is used for matching while synthesising texture. We need to collect a grid of sample points that corresponds to a grid of pixels in the planar patch (Figure 1(a)).

Figure 1 Local resampling: (a) a rectangular neighbourhood pattern on texture sample. Each sample location in the neighbourhood may be reached by tracing an integral path from its adjacent sample point; (b) example of uniform local resampling in the vector field and (c) example of non-uniform local resampling (c) in the same vector field (see online version for colours)



(a) (b) (c)

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 Turk (2001), 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 (Zigelman et al., 2006) 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 synthesised 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 synthesising texture for a small portion of a surface, most of the previous works 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. 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 (Jobard and Lefer, 1997). For each point to be sampled, we draw two straight paths from the centre sample point along the orthogonal integral paths 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 neighbouring 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 1(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.

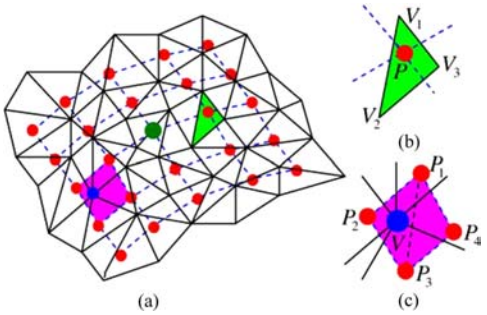
The distance between neighbouring sampling points corresponds to the distance between neighbouring pixels in the texture example. We can sample the points along the integral paths at a uniform distance. The grid-patches obtained by this resampling method illustrate the orientation variation of the vector field but do not reveal the scale variation of the vector field on the surface as shown in Figure 1(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 1(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 synthesise.

When synthesising each patch in the vector field, we need to first extract its neighbourhood 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 2, when extracting neighbour patches, we obtain the colour 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 colour of point X and (w_1, w_2, w_3) is the barycentric coordinates of P in the triangle $\Delta V_1 V_2 V_3$ that contains P.

Figure 2 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) (see online version for colours)



On the other hand, we need to assign colour to the vertices of the mesh when a texture patch is obtained and pasted onto it. As shown in Figure 2(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_2, v_3) 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 colour of the vertex on the mesh can be calculated as follows

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

4 Texture synthesis by optimisation

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

4.1 Texture pre-processing

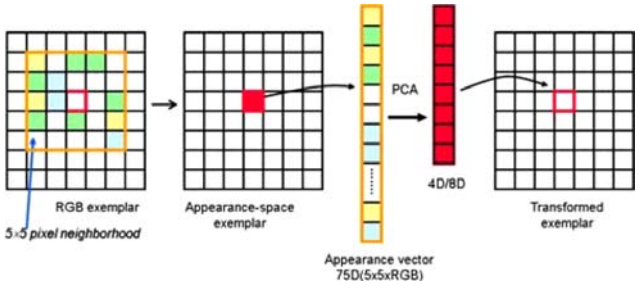
The goal of texture pre-processing is to identify the sets of patches in the exemplar that have similar appearance.

4.1.1 Appearance vector

The traditional approach in texture synthesis is to compare colour neighbourhoods with those of an exemplar (Efros and Leung, 1999; Wei and Levoy, 2000; Efros and Freeman, 2001). Distance is typically measured by summing squared colour differences to compare a synthesised neighbourhood and some exemplar neighbourhood. As is well known, 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 (Lefebvre and Hoppe, 2006), and use L2 distance in appearance-space instead.

As shown in Figure 3, we apply neighbourhood projection (Lefebvre and Hoppe, 2006) to the exemplar itself that is replacing each pixel's RGB colour with the concatenation of all its neighbours' RGB colours. 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 colour is replaced with appearance-space 'colour', which is the so-called appearance vector. Refer to Lefebvre and Hoppe (2006) for more detailed introduction of appearance-space.

Figure 3 Appearance-space is constructed in the same way as Lefebvre and Hoppe (2006) (see online version for colours)



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_I for those patches that are most similar to the current patch B_o, and then one of the most similar patches is used for synthesising.

Actually, we do not need to find the exact n patches, which are most similar to the current patch; we are willing to accept n approximate nearest neighbours. This search can be considered as an n approximate nearest neighbours 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 neighbourhoods we used, and appearance-space technique makes 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 pre-process performs badly for such high dimension, so we apply a special dimension-reduction method before PCA pre-process. Since each appearance vector actually describes the property of a small neighbourhood, we choose some representation pixels and use them to measure the similarity between two texture patches instead of all pixels in the neighbourhoods. After dimension-reduction, efficient searching algorithms (Arya et al., 1998) (i.e., ANN) can be used.

Table 1 Timing comparison between different acceleration methods on Pentium 4 3.0 GHz with 1 G RAM. Texture sample: 96 × 96, patch size: 32 × 32; mesh vertices: 196k; EM iteration: 10; resultant dimension of PCA: 30

Analysis/ synthesis(s)	*Exhaustive*	*PCA + ANN*	*PCA + Clustering*	*Clustering*
Optimisation	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

Inspired by the hierarchical clustering accelerating method proposed in Kwatra et al. (2005), 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 centres 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 synthesising, we first search the optimal cluster centre within all cluster centres, 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 pre-processing step, and the results can be saved for later use.

4.2 Mesh pre-processing

In our approach, we synthesise textures as vertex colours directly over the target mesh surface. Both of our texture synthesis methods need first to obtain all regular-grid patches in the vector fields. The goal of mesh pre-processing 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 centre from which we calculate all the grid-patches with our local resampling method. When progressively variant texture is being synthesised, we select these vertices according to the length of local vectors. More vertices are selected where the local vectors are short. The distance between neighbouring patch centres p is determined by the size of grid-patch. Like Kwatra et al. (2005), we choose former as ¼ latter, so each vertex is covered by four patches on average. All grid-patches are then saved in the resampling patch set Φ. This is done before the synthesis process, to make the texture synthesis fast.

4.3 Texture optimisation

Let X be the vectorised version of the target texture X that we want to synthesise, which is formed by concatenating the colour value of all pixels in X. Let X_p be a subvector of X that corresponds to the neighbourhood of the pixel P. Further, let Z_p be the vectorised pixel neighbourhood in the input sample Z, whose appearance is most similar to X_p. Then, we define our texture energy in a way similar to Kwatra et al. (2005):

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

However, our texture energy is defined in appearance-space instead of RGB-space. The subset $X^\dagger$is obtained from the previous mesh pre-processing step. The coefficient ω_p is used for accelerating the subsequent EM iterative operation, and we set it as suggested in Kwatra et al. (2005):

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

Like Kwatra et al. (2005), we minimise the texture energy E_t with EM-like (Expectation–Maximisation) algorithm.

We first initiate X with random colours. Then, we iterate the EM-algorithm until the texture energy stops changing. In M-Step, we minimise $E_t(X;\{Z_p\})$ 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 minimise $E_t(X;\{Z_p\})$ w.r.t. X. In our implementation of least square solver, we record both colour and weight at each vertex of the mesh. At the beginning of E-Step, both colour 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 colour of each vertex is calculated. When finished synthesising 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 to paste the texture patches of RGB-space to the vector field, and then textures of high quality are obtained. Algorithm 1 describes the pseudocode for texture optimisation algorithm.

Algorithm 1. *Texture Optimization*

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

for iteration n = 0 : N **do**

$X^{n+1} \leftarrow \arg\min_X E_t(X;\{Z_p^n\})$ // E-Step

$Z^{n+1} \leftarrow$ nearest neighbor of X_p^{n+1} in $Z \quad \forall p \in X^\dagger$ // M-Step

if $Z_p^{n+1} = Z_p^n \quad \forall p \in X^\dagger$ **then**

$X \leftarrow X^{n+1}$

break

endif

endfor

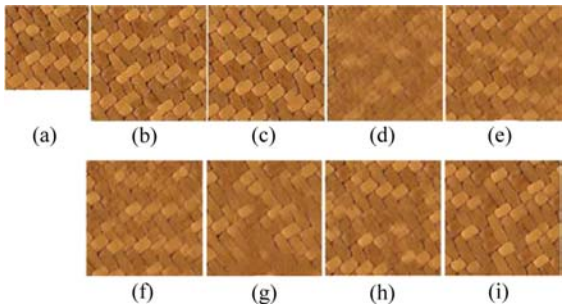
4.4 Multilevel synthesis

To produce higher-quality textures, we adopt a multi-resolution and multiscale fashion similar to Kwatra et al. (2005), Han et al. (2006). 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 synthesising 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 matching.

For the multiscale fashion, we proceed local resampling in the vector field with different neighbourhood sizes, and save the resultant resampling patches during mesh pre-processing. The texture in Figure 4(c) is generated with three resolutions and three neighbourhood sizes. Figure 4(d) shows synthesis at (1/4 resolution, 8×8 neighbourhood). Figure 4(e): (1/2, 16×16). Figure 4(f): (1/2, 8×8). Figure 4(g): (1/1, 32×32). Figure 4(h): (1/1, 16×16). Figure 4(i): (1/1, 8×8). As we can see, multilevel approach obviously produces much better result (in comparison with Figure 4(b)).

Figure 4 Multilevel synthesis: (a) input texture; (b) synthesis result without multilevel; (c) synthesis result with three resolutions and three scales and (d)–(i). Different stages to synthesis (c) (see online version for colours)



5 Texture synthesis by patch-based sampling

As our texture optimisation is quite slow owing to its intrinsic iterative operation, 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 optimisation 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 Soler et al. (2002) and Magda and Kriegman (2003).

Parallelisable: Our approach synthesises texture on surface by pasting patches in a random order and thus achieves parallelism.

Inspired by the work of Liang et al. (2001), and Efros and Freeman (2001), our patch-based sampling approach uses rectangular texture patches of the input sample texture T_I as the building blocks for constructing the synthesised mesh M_o as shown in Figure 5(a). In each step, we paste a patch B_I of T_I onto M_o (Figure 5(c)). Each patch B_I has a boundary zone E_B of width W_E as shown in Figure 5(b). To avoid mismatching features across patch boundaries, we carefully select B_I based on the patches already pasted on M_o. For simplicity, we use only square patches of a prescribed size $\boldsymbol{w}_B \times \boldsymbol{w}_B$.

5.1 Pre-processing

Our patch-sampling approach is also separated into two phases: pre-processing and texture synthesis, and the pre-processing is proceeded in a way similar to previous

optimisation-based approach. In texture pre-processing, we build search space with the boundary of texture patches instead of the whole patch. This is because in patch-based sampling, we need to make neighbouring patches match well at the boundary they have in common, and the inner zones of the texture patches do not affect the synthesis quality. In mesh pre-processing, we choose distance of neighbouring patch centres as ½ patch size. In this way, each point in the vector field is covered by two resampling patches on average, but the total number of resampling patches is much smaller than optimisation approach, so making this synthesising 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 colour to the vertex.

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

Step 3: Select a patch B_I from T_I such that its boundary is the closest to that of B_o.

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

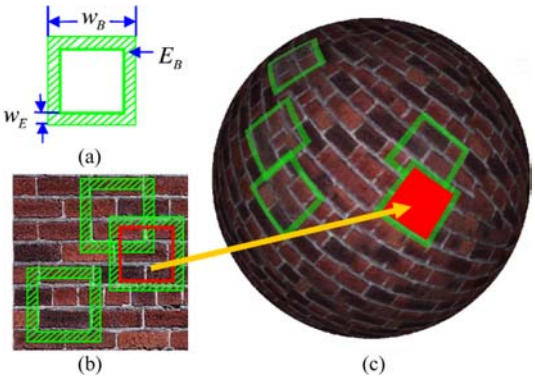
Step 5: Repeat steps 2–4 until all the vertices of M_o are coloured.

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

The synthesis process starts by randomly colouring 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 neighbours in the mesh. We continue until the whole surface is covered.

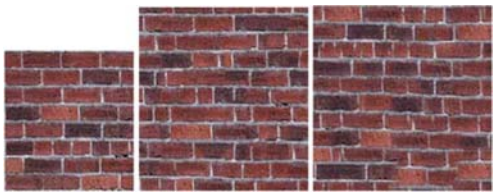
In Liang et al. (2001) and Efros and Freeman (2001), texture is synthesised in scanline order, for each patch to be synthesised there are always one or two neighbouring patches synthesised earlier, then one can easily select a patch from the exemplar and make it match well with its neighbouring patches in their common boundary. However, our approach succeeds by adequate 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 three-level hierarchies to synthesise the texture. Three-level hierarchy works well for most of the examples in our experiments.

Figure 5 Patch-based sampling strategy: (a) the $w_B \times w_B$ patch B_I has a boundary zone E_B of width W_E; (b) the input texture sample and (c) the synthesised texture on surface. In the input texture sample shown in (b), three patches have boundary zones matching the red texture patch B_I shown in (c) and the red patch is selected (see online version for colours)



Our approach synthesises the texture over the resampling patch set in random order. Note that we can also synthesise the texture over the patch set in a breadth-first order, which is similar to Liang et al. (2001), and Efros and Freeman (2001); in each step we try to find a patch synthesised earlier and synthesise its untextured neighbouring 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 Liang et al. (2001) and our approach. Figure 6 shows an example comparing the results.

Figure 6 Texture synthesis results comparison: (a) input texture; (b) synthesis result using the approach of Liang et al. (2001) and (c) synthesis result using our approach with random order. For most textures, we have found that random-order works well (see online version for colours)



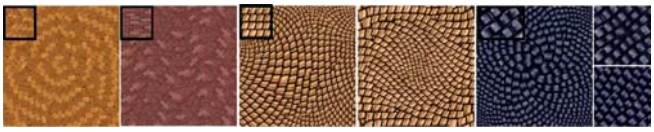
6 Experimental results

We have tested both of our texture synthesis algorithms on a variety of textures and models. Parameters are important for synthesising texture. The size of the texture patch affects how well the synthesised texture captures the local characteristics of the texture exemplar. For texture optimisation, we use three resolution levels and successive

neighbourhood 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 neighbourhood's size, and other resolutions and neighbourhood 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 7, we show several examples for synthesising different textures over different planar vector fields, and compare them with the results from some other techniques. Note that different from Kwatra et al. (2005), 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 changes over the vector fields. Our results look quite good and preserve the orientation of the vector fields, and scale variation is revealed. Comparing (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 7 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 Kwatra et al. (2005); (e) non-uniform texture over the ring-shaped vector field and (f) results from Kwatra et al. (2005) (upper) and Lefebvre and Hoppe (2006) (see online version for colours)



(a) (b) (c) (d) (e) (f)

Figure 8 shows the results of uniform and non-uniform texture synthesis over the bunny mesh model. Note that the progressively variant texture (Figure 8(a)) has the same orientation with the uniform texture result (Figure 8(b)), but differs in the scale variation. Figure 9 compares our synthesis result with that from Lefebvre and Hoppe (2006). It is 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 produces better quality by comparing the results near the tail and left ear of the bunny model.

Figure 8 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 (see online version for colours)

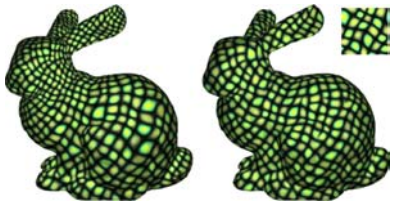


Figure 9 Comparison of texture synthesis on the bunny model: (a) result of Lefebvre and Hoppe (2006) and (b) our result (see online version for colours)

(a) (b)

Different textures are synthesised 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 optimisation, while the rest are generated with patch-based sampling.

Figure 10 Synthesising different textures on a head model (see online version for colours)

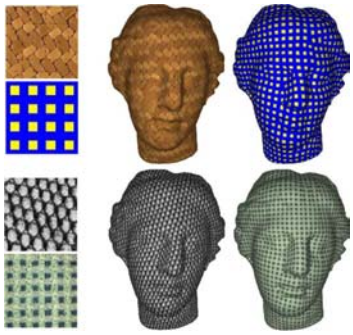


Figure 11 Some texture synthesis results for different meshes and textures (see online version for colours)

The texture optimisation 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 artefacts on the edges of patches for some texture. However, the latter is much faster.

Table 1 summarises 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.

7 Conclusions

To synthesise distortionless texture on surfaces, we present an intuitive and intrinsic local resampling algorithm in vector fields, and extend both texture optimisation and patch-based sampling texture synthesis algorithms to 2D 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 synthesising 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 Han et al. (2006), and Kwatra et al. (2005), least square solver will introduce blurry blending problem for texture optimisation. On 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 works is further acceleration of our texture synthesis methods, especially the texture optimisation approach. Note that both of our texture synthesis methods are proceeded in random order, so it should not be too complicated to implement our methods on modern hardware.

Acknowledgement

This work is supported by the joint grant of the National Natural Science Foundation of China and Microsoft Research Asia (60776799) and the 973 National Key Basic Research Foundation of China (No. 2009CB320801).

References

Arya, S., Mount, D., Netanyahu, N., Silverman, R. and Andwu, A. (1998) 'An optimal algorithm for approximate nearest neighbour searching', *Journal of ACM*, Vol. 45, pp.891–923.

Ashikhmin, M. (2001) 'Synthesising natural textures', *Proc. Symposium on Interactive 3D Graphics*, New York, pp.217–226.

Cohen, M., Shade, J., Hiller, S. and Deussen, O. (2003) 'Wang tiles for image and texture generation', *Proc. SIGGRAPH*, San Diego, California, pp.287–294.

Dischler, J., Maritaud, K., Levy, B. and Ghazanfarpour, D. (2002) 'Texture particles', *Journal Computer Graphics Forum*, pp.401–410.

Efros, A. and Freeman, W. (2001) 'Image quilting for texture synthesis and transfer', *Proc. SIGGRAPH*, New York, NY, USA, pp.341–346.

Efros, A. and Leung, T. (1999) 'Texture synthesis by non-parametric sampling', *International Conference on Computer Vision*, Vol. 2, No. 9, pp.1033–1038.

Fu, C-W. and Leung, M-K. (2005) 'Texture tiling on arbitrary topological surfaces', *Proc. Eurographics Symposium on Rendering*, Konstanz, Germany, pp.99–104.

Han, C., Risser, E., Ramamoorthi, R. and Grinspun, E. (2008) 'Multiscale texture synthesis', *Proc. SIGGRAPH*, New York, NY, USA, pp.1–8.

Han, J., Zhou, K., Wei, L-Y., Gong, M., Bao, H., Zhang, X. and Guo, B. (2006) 'Fast example-based surface texture synthesis via discrete optimisation', *Visual Computer*, pp.918–925.

Jobard, B. and Lefer, W. (1997) 'Creating evenly-spaced streamlines of arbitrary density', *Proc. Eurographics Workshop on Visualization in Scientific Computing*, pp.45–55.

Kopf, J., Fu, C., Cohen-or, D., Deussen, O., Lischinski, D. and Wong, T. (2007) 'Solid texture synthesis from 2D exemplars', *Proc. SIGGRAPH*, New York, NY, USA, p.2.

Kwatra, V., Adalsteinsson, D., Kim, T., Kwatra, N., Carlson, M. and Lin, M. (2007) 'Texturing Fluids', *IEEE Transactions on Visualisation and Computer Graphics*, Vol. 13, No. 5, pp.939–952.

Kwatra, V., Essa, I., Bobick, A. and Kwatra, A. (2005) 'Texture optimization for example-based synthesis', *Proc. SIGGRAPH*, New York, NY, USA, pp.795–802.

Kwatra, V., Schodl, A., Essa, I., Turk, G. and Bobick, A. (2003) 'Graphcut textures: image and video synthesis using graph cuts', *Proc. SIGGRAPH*, New York, NY, USA, pp.277–286.

Lefebvre, S. and Hoppe, H. (2006) 'Appearance-space texture synthesis', *Proc. SIGGRAPH*, New York, NY, USA, pp.541–548.

Liang, L., Liu, C., Xu, Y., Guo, B. and Shum, H. (2001) 'Real-time texture synthesis by patch-based sampling', *ACM Transactions on Graphics*, Vol. 20, No. 3, pp.127–150.

Magda, S. and Kriegman, D. (2003) 'Fast texture synthesis on arbitrary meshes', *Proc. Eurographics Symposium on Rendering*, Leuven, Belgium, pp.82–89.

Nealen, A. and Alexa, M. (2003) 'Hybrid texture synthesis', *Proc. Eurographics Symposium on Rendering*, Leuven, Belgium, pp.97–105.

Praun, E., Finkelstein, A. and Hoppe, H. (2000) 'Lapped textures', *Proc. SIGGRAPH*, New York, NY, USA, pp.465–470.

Soler, C., Cani, M. and Angelidis, A. (2002) 'Hierarchical pattern mapping', *Proc. SIGGRAPH*, New York, NY, USA, pp.673–680.

Takayama, K., Okabe, M., Ijiri, T. and Igarashi, T. (2008) 'Lapped solid textures: filling a model with anisotropic textures', *Proc. SIGGRAPH*, New York, NY, USA, pp.1–9.

Tong, X., Zhang, J., Liu, L., Wang, X., Guo, B. and Shum, H. (2002) 'Synthesis of bidirectional texture functions on arbitrary surfaces', *Proc. SIGGRAPH*, San Antonio, Texas, pp.665–672.

Turk, G. (2001) 'Texture synthesis on surfaces', *Proc. SIGGRAPH*, New York, NY, USA, pp.347–354.

Wei, L. and Levoy, M. (2000) 'Fast texture synthesis using tree structured vector quantisation', *Proc. SIGGRAPH*, New York, NY, USA, pp.479–488.

Wei, L. and Levoy, M. (2001) 'Texture synthesis over arbitrary manifold surfaces', *Proc. SIGGRAPH*, New York, NY, USA, pp.355–360.

Wei, L., Han, J., Zhou, K., Bao, H., Guo, B. and Shum, H. (2008) 'Inverse texture synthesis', *Proc. SIGGRAPH*, New York, NY, USA, pp.1–9.

Xu, Y., Guo, B. and Shum, H. (2000) *Chaos Mosaic: Fast and Memory Efficient Texture Synthesis*, Technical Report MSR-TR-2000-32 of Microsoft Research.

Ying, L., Hertzmann, A., Biermann, H. and Zorin, D. (2001) 'Texture and shape synthesis on surfaces', *Proc. Eurographics Symposium on Rendering*, London, UK, pp.301–312.

Zelinka, S. and Garland, M. (2003) 'Interactive texture synthesis on surfaces using jump maps', *Proc. Eurographics Symposium on Rendering*, Leuven, Belgium, pp.90–96.

Zhang, J., Zhou, K., Velho, L., Guo, B. and Shum, H-Y. (2003) 'Synthesis of progressively variant texture on arbitrary surfaces', *Proc. SIGGRAPH*, San Diego, California, pp.295–302.

Zigelman, G., Kimmel, R. and Kiryati, N. (2002) 'Texture mapping using surface flattening via multidimensional scaling', *IEEE Transactions on Visualisation and Computer Graphics*, Vol. 8, No. 2, pp.198–207.