# Synthesizing 2D Directional Moving Texture

Bin Wang*
Tsinghua University/Alice-ISA, Loria

Wenping Wang†
The University of Hong Kong

Junhai Yong‡
School of Software, Tsinghua University

Jiaguang Sun§
School of Software, Tsinghua University

## Abstract

We present a novel patch-based algorithm for synthesizing a moving 2D texture, i.e. a sequence of frame-coherent 2D textures. In our method, the input are a sample texture and a 2D flow field. We first synthesize a 2D directional texture according to the direction information of the flow field and then let the texture move following the flow. Iteratively, the texture $T_{i+1}$ of the $(i+1)$-th frame is first obtained by moving forward the texture $T_i$ in a piecewise manner. Then necessary hole-filling and blending is used to make $T_{i+1}$ coherent with $T_i$. In addition, to maintain good visual quality throughout the sequence of textures, best-matching patches from the sample texture are used at selected locations of $T_{i+1}$ to prevent cumulative blurring due to blending. Our test examples show that our method is capable of generating high quality moving textures with attractive visual effects that are useful for flow visualization.

**CR Categories:** I.3.3 [Computer Graphics]: Picture/Image Generation— [I.4.7]: Image Processing and Computer Vision— Feature Measurement–Texture

**Keywords:** vector field, moving texture, texture synthesis

## 1 Introduction

Texture synthesis is a technique that uses a sample texture to generate a synthesized texture that is visually similar to the sample texture but not a verbatim copy. Texture synthesis has attracted active research efforts due to wide application of textures in appearance modeling in computer graphics. All existing methods for texture synthesis consider the synthesis of static textures. We present in this paper the first method to synthesize a moving texture, i.e a sequence of frame-coherent 2D synthesized textured images, from a static sample texture and a flow field. Besides the common issues encountered in synthesizing static textures, the new challenges in our work are to ensure coherence between consecutive frames and to maintain good visual clarity of a moving texture.

*e-mail:bin.wang@loria.fr
†e-mail:wenping@cs.hku.hk
‡e-mail:yongjh@tsinghua.edu.cn
§e-mail:sunjg@tsinghua.edu.cn



Among existing methods for texture synthesis, pixel-based methods and patch-based methods are relatively recent developments and are most relevant to the present paper. The papers [Efros and Leung 1999], [Hertzmann et al. 2001], [Tong et al. 2002], [Turk 2001], [Wei and Levoy 2000], [Wei and Levoy 2001], [Ying et al. 2001] and [Gorla et al. 2003] belong to the category of the pixel-based approach; among them, some papers focus on 2D texture synthesis and the others are about texture synthesis over surfaces. Patch-based texture synthesis methods have proven faster than pixel-based methods. Xu et al. [Xu et al. 2000] use square patches of fixed size randomly selected from the sample texture and alpha-blend the overlapping parts of adjacent patches to synthesize a texture. Xu et al.'s method is improved by Efros and Freeman [Efros and Freeman 2001] stitching together adjacent texture patches along a minimum-error boundary in their overlapping region. Liang et al. [Liang et al. 2001] extend Xu et al.'s method by finding the patch in the overlapping region which most closely resembles the adjacent patches already synthesized, and compositing the new patch with those already synthesized using feather blending. Wang et al. [Wang et al. 2004] extend Liang et al's method to handle texture patches of different sizes and orientations and to synthesize directional textures in regions with irregular boundaries.

In this paper, we study here how to generate a *moving texture*, i.e. a sequence of frame-coherent directional textures, i.e. an animation sequence of textures. There are two requirements for such a moving texture. First, the structure of the sample texture should be preserved and the direction of the flow field should be reflected in each frame of texture. Second, consecutive frames should be coherent, i.e. continuous over time. Given a steady flow field and a small sample texture, we first use the patch-based method presented in [Wang et al. 2004] to synthesize a large directional texture, as the first frame, according to the direction of the flow field. Take Figure 10 as an example of this step. Figure 10(a) is a sample texture of canes; Figure 10(b) is a flow field in which vectors at all pixels are tangent to a family of concentric circles. Figure 10(c) is a synthesized directional texture, which is the first frame of an animation sequence showing the flow defined in Figure 10(b). Clearly, Figure 10(c) shows the directions of the flow field, while maintaining the structure of the sample texture.

Once the first frame $T_1$ has been obtained, following the velocity of the flow field, we move properly subdivided cells in $T_1$ to obtain the texture $T_2$ of the next frame. Hole-filling and blending operations are applied to make $T_2$ seamless and coherent with the previous frame $T_1$. The above step is applied iteratively to generate the texture $T_{i+1}$ of the $(i+1)$-th frame from $T_i$, $i > 0$. In this way, the frame coherence between consecutive frames is preserved, while the texture of each frame resembles the sample texture. The sequence of textures thus generated produces an animation of a moving texture.

Our method bears some similarity to previous texture advection methods ([van Wijk 2002], [Neyret 2003]) developed for flow visualization. These methods, although fast, are only applicable to random noise images and cannot be used with structured textures. We take a step beyond this status quo to consider advecting a structured moving texture along a 2D flow field.

## 2 Algorithm

Our algorithm takes a sample texture and a flow field as input, and generates a sequence of frame-coherent textures which move following the flow field. The following are the main steps in our method:

(1) The texture $T_1$ of the first animation frame is synthesized. Set the current frame $T_{ct}$ to be the first frame $T_1$.

(2) Subdivide $T_{ct}$ into cells that observe the velocity constraint that all pixels in each cell share similar flow velocity. Move these cells according to the flow field to get a texture $T_{nt}$ for the next frame.

(3) Since the moved cells from $T_{ct}$ may overlap or leave some pixels unfilled, creating *holes*, in $T_{nt}$, we use blending and hole-filling operations to repair $T_{nt}$ to make $T_{nt}$ continuous, while keeping it coherent with $T_{ct}$.

(4) To prevent texture visual quality from deteriorating due to cumulative effect of blending in Step (3), we use best-matching patches from the sample texture to re-synthesize regions that have gone through a certain number of blending operations.

(5) If the last frame has been reached, stop; otherwise, set the next frame as the current frame, i.e. $T_{ct} := T_{nt}$ and go to Step (2).

The above steps are elaborated in the following subsections.

### 2.1 Synthesizing the first frame

We explain now how to synthesize the first frame as a 2D directional texture, following [Wang et al. 2004]. We first subdivide the given vector field into cells of the same size and then check the directions of all the pixels within each cell. If the directions do not satisfy the *direction constraint*, the cell is then subdivided into four smaller cells. And we then recursively check the four smaller cells to see whether they should be subdivided or not. The direction constraint in our implement is set to $|a_i - a_j| < 2\pi/n$ for all $i, j$, where $i, j$ are pixels within the cells and $a_i$ is the direction on the pixel $i$; $n$ is typically set to 24 or 48. Furthermore, to avoid efficiency degradation, we limit the cell size to be $4 \times 4$ pixels at least, i.e. only cells larger than $4 \times 4$ can be subdivided. We observed that, for most vector fields, using $4 \times 4$ pixels as the smallest cell size strikes a good balance between keeping the efficiency of patch-based synthesis and accommodating the direction variation of a vector field. The largest cell size is set to be $32 \times 32$. Figure 1 shows many cells over a vector field to be synthesized.

All cells in the first frame are synthesized from left to right and bottom to top. We use Figure 2 to briefly illustrate this procedure, which is discussed in detail in [Wang et al. 2004]. Figure 2(a) gives a sample texture. Suppose that the cell $\alpha$, also called a *patch*, in Figure 2(b) is the next cell to be synthesized. Since the direction of the vector field at the cell $\alpha$ is not parallel to the horizontal side of $\alpha$, we search in a rotated copy of the sample texture (Figure 2(c)) to find a best-fit patch $\gamma$, and paste $\gamma$ at the cell $\alpha$. The "best fit" means that the L-shaped boundary (white shadow area) of $\alpha$ and the L-shaped boundary of $\gamma$ have minimum difference. To further eliminate boundary artifacts, in a preprocessing step, we use feather blending within the boundaries of the neighboring patches as in [Liang et al. 2001] and [Wang et al. 2004]. To speed up the search for a best-fit patch in a rotated copy of the sample texture, we compute and store $n$ ($n = 24$ or 48) rotated copies of the sample texture for efficient access later. The main direction of the sample texture is determined using the algorithm of Tamura et al. [Tamura et al. 1978].

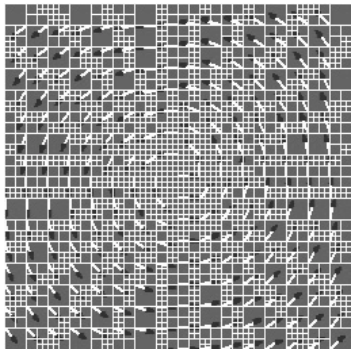


Figure 1: The cells to be synthesized after the preprocess stage.

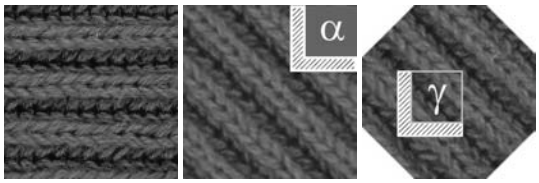



Figure 2: Synthesis of a 2D directional texture.

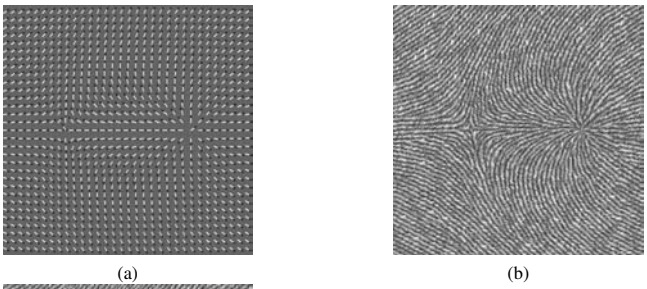
### 2.2 Synthesizing a new frame

Suppose that the first frame has been synthesized. Now we consider how to synthesize the subsequent frames.

#### 2.2.1 Moving patches

The synthesis of the texture of a new frame from its previous frame is the most critical, and also the most difficult task, since one needs to ensure both frame coherence and visual quality of the new frame. A straightforward but naive approach would be to move all pixels in the previous frame following the vector field to generate the texture for the next frame. However, with this approach, the texture structure would be destroyed gradually by the accumulative stretching or shrinking due to the non-uniform velocity of the vector field, as illustrated by Figure 3. Figure 3(a) shows a vector field, Figure 3(b) shows the first frame of moving texture, and Figure 3(c) shows the 20-th frame of animation generated by consecutive movement of individual pixels. It is evident that the original structure of the sample texture is totally lost in a large portion of the 20-th frame.

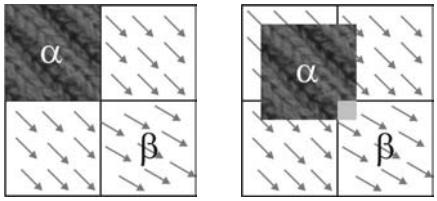
To circumvent the difficulty of keeping the texture structure while letting the texture move following the flow field, we take the following approach again based on cell subdivision. We subdivide the current frame into cells that satisfy the *velocity constraint* that the velocities of all the pixels in the same cell do not differ much, i.e. below some threshold (for example, twice of the minimal velocity in the cell). A cell is further subdivided into four smaller cells if its size is larger than $4 \times 4$ *and* the velocity constraint is not satisfied. Then we move every cell to a new position according to its average speed.



(a) (b)

(c)

Figure 3: An illustration of the effect of moving pixels individually: (a) A vector field; (b) The first frame of moving texture; (c) The 20-th frame of moving texture.




(a) (b)

Figure 4: An illustration of moving patches and validation of pixels.

### 2.2.2 Checking validity of pixels

Some pixels in the next frame may not be covered by any moved cells from the current frame or may be covered by more than one moved cell. These pixels are marked as *invalid*. Furthermore, for a pixel covered only by one cell, if the flow vector direction at the pixel differs from the average texture direction of the covering cell by more than a threshold, this pixel will also be marked as invalid. Figure 4 illustrates the process of checking for invalid pixels. The patch $\alpha$ in Figure 4(a) is moved to its new location in Figure 4(b) according to its average speed. However, since the flow directions in cell $\beta$ are not consistent with the texture directions in $\alpha$, the pixels in the small square (light gray, on the up-left corner of $\beta$) in Figure 4(b) are marked as invalid. Invalid pixels form holes that need to be filled by a subsequent synthesis process.

### 2.2.3 Filling holes

Figure 5 shows an example image with "holes" shown in black. There are two requirements for filling the holes. The first is that the filling texture be joined seamlessly with its surrounding texture. The second is that the direction of the filling texture follow the vector directions of the pixels in the holes. To deal with irregular hole boundaries, we use image masks to search for the best-fit filling patches in the pre-computed rotated copies of the sample texture. To build an image mask, we first define a rectangular cell which covers the hole, as shown in Figure 6 and check every valid pixel

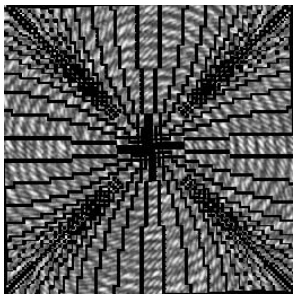
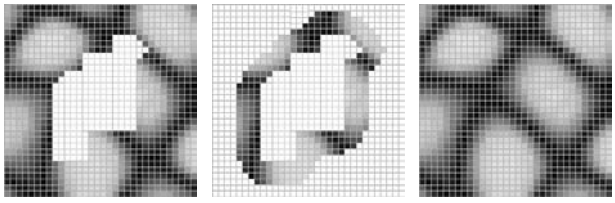


Figure 5: Many holes, shown as black regions, result from by validity checking.



a b a

Figure 6: (a) A cell with a hole (the white region); (b) The image mask for filling the hole. (c) The hole is filled

in the cell. If the distance from the pixel to the hole boundary is within a threshold $D$ ($D = 4$, typically), the pixel is in the image mask; otherwise, the pixel is not in the image mask. In this way, for example, we construct the image mask in Figure 6(b) for the hole in Figure 6(a). If a cell covering a hole is larger than the largest size limit($32 \times 32$) or the pixels in the hole do not satisfy the *direction constraint*, we subdivide the cells into four smaller cells and build mask for every cell.

To produce a smooth joining between the filling patch and the surrounding patches of a hole, we define a blending function over the image mask by

$$B(d) = \begin{cases} 0, & d > D, \\ 1 - d/(D+1), & 0 < d \le D, \end{cases} \tag{1}$$

where $d$ is the distance from a pixel to the hole boundary and $D$ is the distance threshold, i.e. the width of the image mask (see Figure 6(b)). Aided by the image mask, we search in an appropriate rotated copy of the sample texture to find the best-fit patch to fill the hole. The fitting error is defined as

$$E = \sum_{i=0}^{B_1-1} \sum_{j=0}^{B_2-1} T(i,j)(F(i,j) - G(i+u, j+v))^2, \tag{2}$$

where $T(i,j)$ is the characteristic function of the support binary image of the image mask, i.e. $T(i,j) = 1$ if the pixel $(i,j)$ is in the mask, and $T(i,j) = 0$ otherwise; $B_1$, $B_2$ are cell widths, $F(i,j)$ is the pixel value in the image mask, $G(i,j)$ represents the sample texture, and $(u,v)$ is the left-bottom corner of a patch with the size $B_1 \times B_2$ in the sample texture. Then, for all such patches in the sample texture, we choose the one with the minimal $E$ as the best-fit patch, and use it to fill the hole and blend it with the surrounding patches using the blending function given by Equation 1.

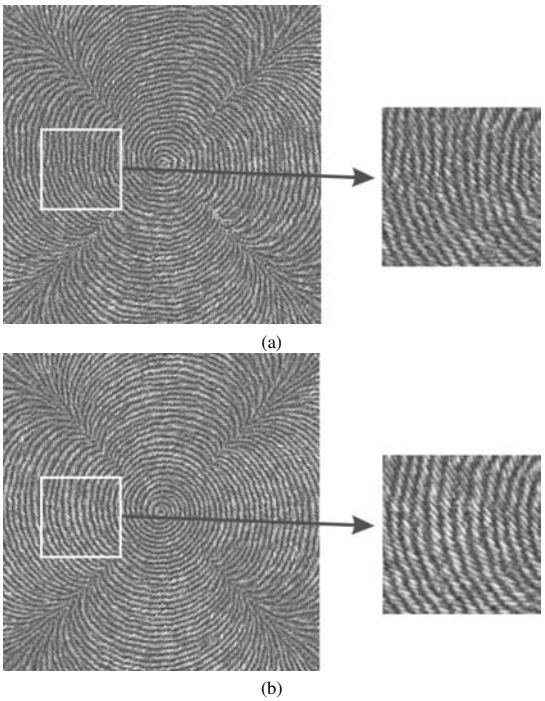


Figure 7: (a) Result without removing the blurring effect; (b) Result after removing the blurring effect.

### 2.3 Removal of blurring

Since blending is used repeatedly for hole-filling, if unchecked, its cumulative blurring effect would degrade the visual quality of the moving texture towards the end of animation. To address this problem, we keep the count of the number of times blending has been applied to a particular pixel and we check the average blending count of each cell. If the average count of a cell is larger than certain threshold, which is typically set to 2, the cell will be marked as *over-blended* and will be re-synthesized by a best-fit patch from the sample texture. The procedure of searching for the best-fit patch is similar to that in hole filling, however, the error is not defined within the boundary but in the whole cell. This remedy step proves to be effective in maintaining the image quality of the moving texture, as it provides a safeguard against potentially blurring due to repeated hole-filling and blending operations. Take Figure 7 for example. Figure 7(a) shows the 100-th frame of a sequence of textures without removing the blurring effect, and Figure 7(b) shows the 100-th frame of the same sequence but with the removal of the blurring effect. The latter has better image quality than the former as shown by the zoomed-in views in Figure 7.

## 3 Results and Discussion

Figures 8 to 11 show four moving texture examples. In every figure, (a) shows the input sample texture and (b) shows the steady flow field. (The vector field of Figure 11 is the same as the one in Figure 3(a)). The four frames in Figure 8 through Figure 11 are the first, the 30-th, the 70-th and the 100-th frames of the synthesized moving textures. We see that the image quality is preserved very well throughout the animation and the texture of each frame resembles the sample texture. In Figures 8, 9 and 11, we use normalized velocity for every pixel. However, in Figure 10, each pixel has the same angular velocity, that means the speed of every pixel is proportional to its distance to the circle center. Notice that the velocity of the pixel near the center is very small and the velocity near the boundary of the circle is very large. This great velocity difference may cause the texture near the boundary to move too fast, while the texture near the center moves too slow or even does not move because the distance of every step is smaller than one pixel. To avoid these undesirable effects of possible aliasing, we limit the velocity of every pixel within a range while moving the patches to make sure that the texture cannot move too fast or too slow. We use an accumulated buffer to record the total moving distance of each slow moving pixel, and actually move a "slow" pixel only when its accumulated distance is beyond a threshold.

Our algorithm produces satisfactory results for many textures, but not all. In general, semi-regular and anisotropic textures tend to produce the best results. For example, the isotropic sample texture in (Figure 12(a)) produces the first frame in (Figure 12(c)) that does not exhibit any directional property of the vector field. Figure 12(d) shows a sample texture with which our algorithm does not work well. This sample texture leads to unacceptable visual artifacts in the 20-th frame of animation (sees Figure 10(b) for the flow field used in this example). The reason for this failure is that the direction constraint in the small cells at the central region does not respect the large brick structure very well.

## 4 Conclusion and Future Work

We have presented a novel method for synthesizing a moving texture following a steady flow. Our method is the first for synthesizing a moving structured texture and provides a useful technique for generating special visual effects in computer graphics. Besides exploiting the latest techniques for synthesizing a 2D directional texture, we have successfully addressed the problem of preserving the frame coherence of the moving texture, while maintaining the original texture structure and visual quality of each texture frame. Furthermore, unlike most previous methods that use only random noise textures for texture advection, our method allows the use of more regular and structural texture, thus enhancing greatly the visualization effects by using texture structures to communicate more clearly the streamlines of a flow field.

There are some obvious problems that call for further research. With our current implementation it takes about 2 minutes on a common PC to generate a new texture frame, with the most time spent on hole-filling. So one problem is how to improve the efficiency of our method or devise other faster methods for solving the same problem of synthesizing a moving texture for a given flow field, hopefully achieving real-time performance. Also, we note that all the examples presented here use steady flow fields. We expect our method to work for an unsteady flow as well, as long as the flows do not change radically with respect to time. Finally, it is a natural extension to consider synthesizing a moving texture over a surface in 3D space.

## References


EFROS, A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *Proc. SIGGRAPH '01*, 341–346.

EFROS, A., AND LEUNG, T. 1999. Texture syntheis by non-parametric sampling. In *Proc. IEEE International Conference on Coupter Vision(ICCV'99)*, 1033–1038.

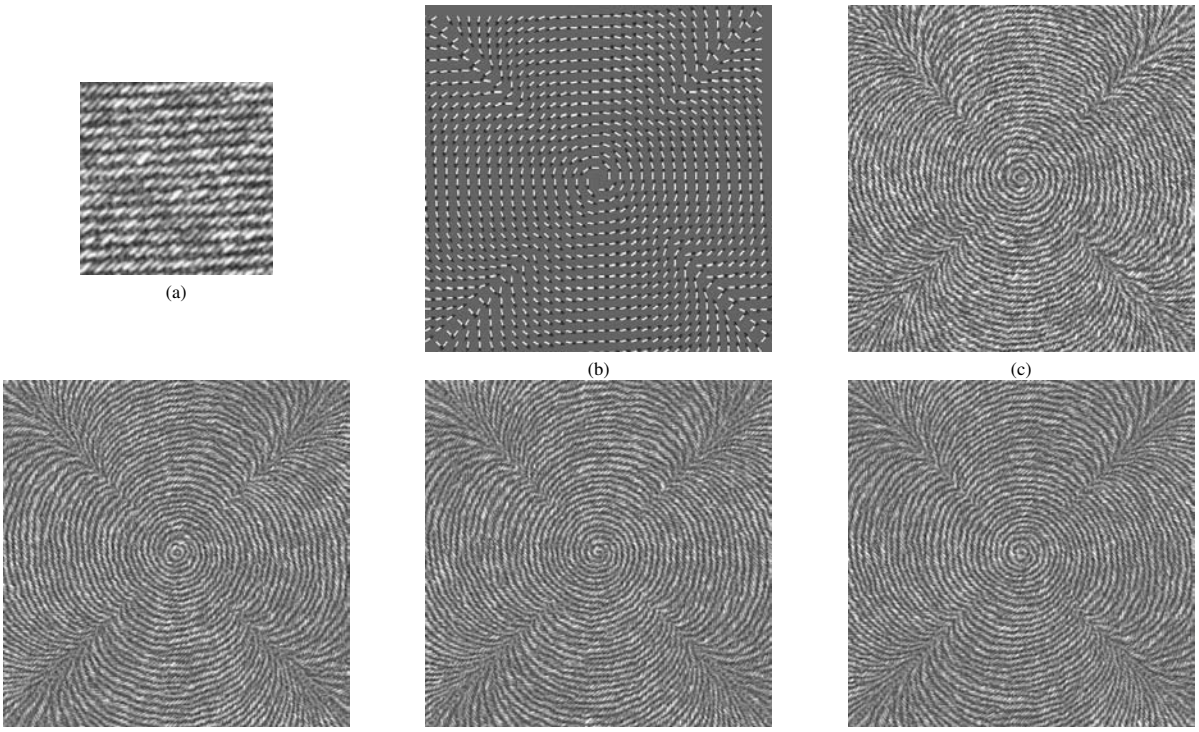



Figure 8: (a) An input sample texture (size $128 \times 128$); (b) A flow field; (c) The 1st frame of the animation (size $512 \times 512$); (d) The 30th frame of the animation; (e) The 70th frame of the animation; (f) The 100th frame of the animation.

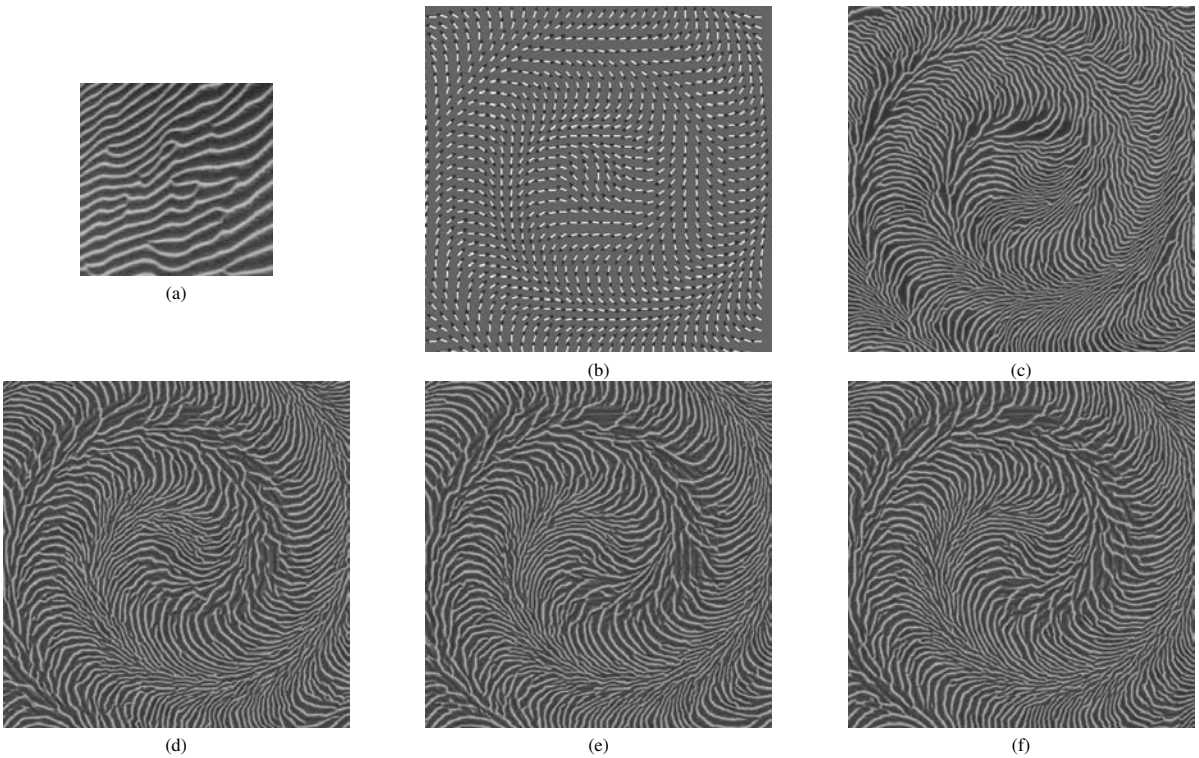



Figure 9: (a) An input sample texture (size $128 \times 128$); (b) A flow field; (c) The 1st frame of the animation (size $512 \times 512$); (d) The 30th frame of the animation; (e) The 70th frame of the animation; (f) The 100th frame of the animation.

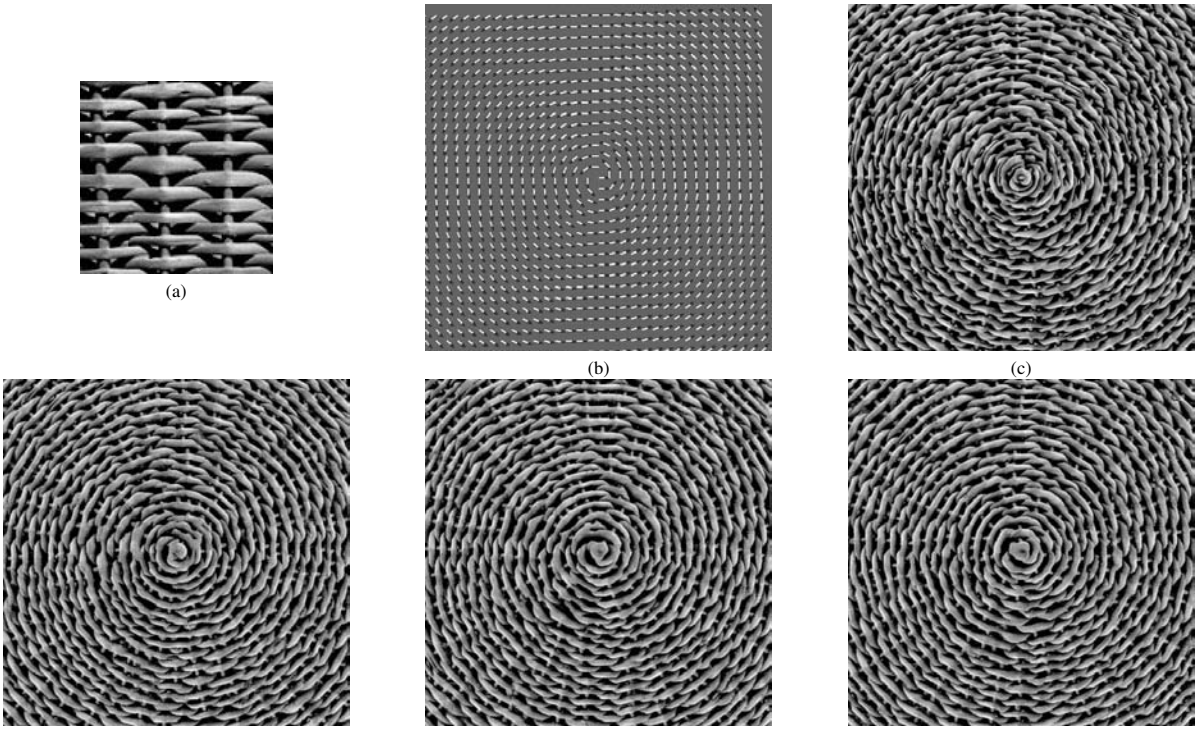



Figure 10: (a) An input sample texture (size $128 \times 128$); (b) A flow field; (c) The 1st frame of the animation (size $512 \times 512$); (d) The 30th frame of the animation; (e) The 70th frame of the animation; (f) The 100th frame of the animation.

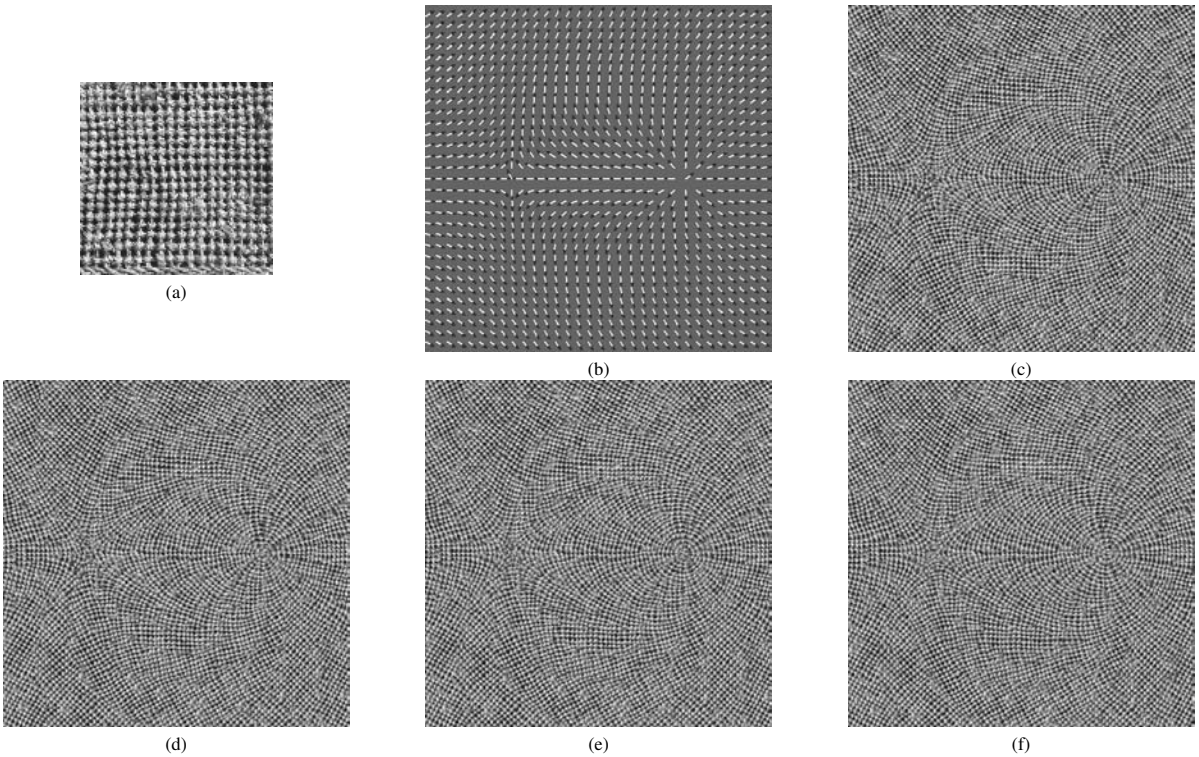



Figure 11: (a) An input sample texture (size $128 \times 128$); (b) A flow field; (c) The 1st frame of the animation (size $512 \times 512$); (d) The 30th frame of the animation; (e) The 70th frame of the animation; (f) The 100th frame of the animation.

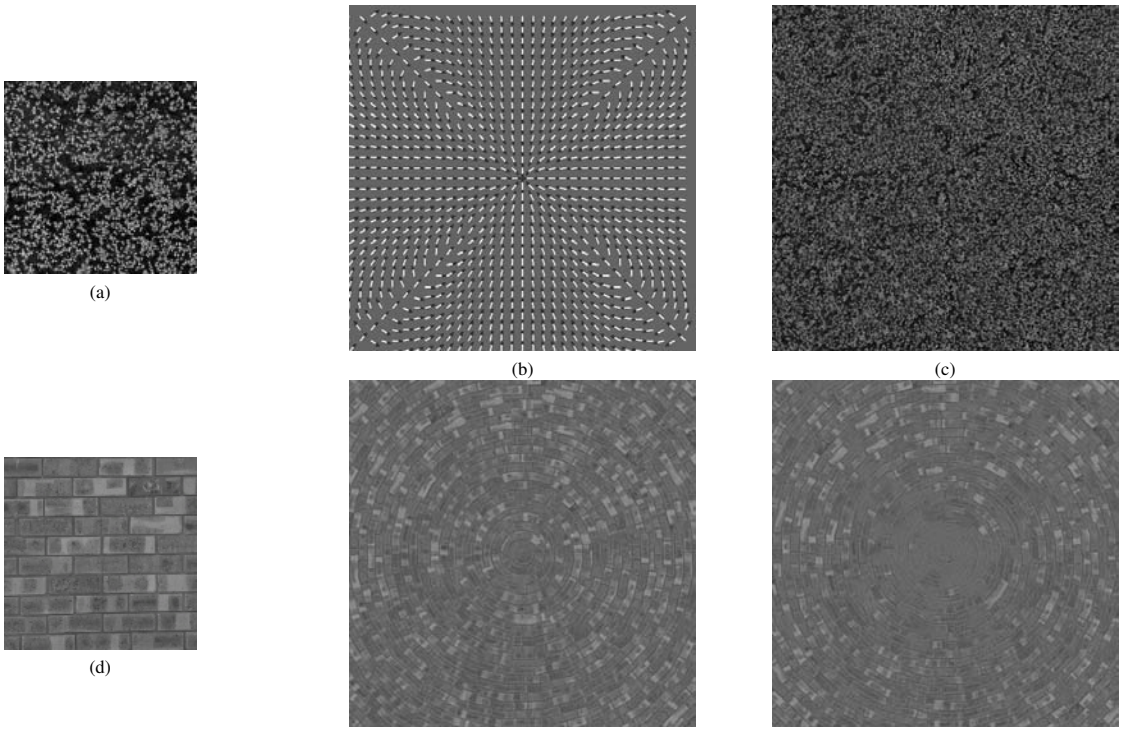



Figure 12: Failed examples: (a) An isotropic sample texture (size $128 \times 128$); (b) A flow field; (c) The first frame of the animation; (d) A sample texture of brick (size $128 \times 128$); (e) The first frame of animation; (f) The 20th frame of animation.


GORLA, G., INTERRANTE, V., AND SAPIRO, G. 2003. Texture synthesis for 3d shape representation. *IEEE Trans. Vis. Comput. Gr. 9*, 4, 212–224.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proc. SIGGRAPH '01*, 327–340.

LIANG, L., LIU, C., XU, Y. Q., GUO, B., AND SHUM, H. Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM Trans. Gr. 20*, 3 (July), 127–150.

NEYRET, F. 2003. Advected textures. In *Eurographics/SIGGRAPH Symposium on Computer Animation'03*.

TAMURA, H., MORI, S., AND YAMAWAKI, T. 1978. Textural features corresponding to visual perception. *IEEE Trans. Syst. Man Cyber. 8*, 6, 460–472.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H. Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Trans. Gr. 21*, 3 (July), 665–672.

TURK, G. 2001. Texture synthesis on surfaces. In *Proc. SIGGRAPH '01*, 347–354.

VAN WIJK, J. J. 2002. Image based flow visualization. *ACM Trans. Gr. 21*, 3 (July), 745–754.

WANG, B., WANG, W. W., YANG, H. P., AND SUN, J. G. 2004. Efficient example-based painting and synthesis of 2d directional texture. *IEEE Trans. Vis. Comput. Gr. 10*, 3, 266–277.

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

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

XU, Y. Q., GUO, B., AND SHUM, H. Y. 2000. Chaos mosaic: Fast and memory efficient texture synthesis. In *Tech. Rep. 32, Microsoft Research Asia*.

YING, L. X., HERTZMANN, A., BIERMANN, H., AND ZORIN, D. 2001. Texture and shape synthesis on surfaces. In *Proc. of 12th Eurographics Workshop on Rendering*, 301–312.