A NOVEL ALGORITHM FOR SYNTHESIZING DIRECTIONAL TEMPORAL TEXTURE ANIMATION

Chao-Hung Lai     Jiunn-Lin Wu
Department of Computer Science,
National Chung Hsing University,
Taichung 402, Taiwan, ROC.
{phd9415 | jlwu}@cs.nchu.edu.tw

ABSTRACT
In this paper, we propose a novel algorithm for directional temporal texture synthesis. The generated temporal textures can move in any user-specified direction at run time, while it requires only a static texture image as input. We first synthesize texture sequences that approximate to true video clips by a texture sequence synthesis algorithm. Then we use transition probabilities to generate infinite length sequences with semi-regular characteristic. The quality of synthesized temporal textures can be improved via cross-fading technique. We also extend our algorithm to interactive rendering. It allows users to design the image masks and warp grids by using mouse drag-and-drop. The temporal textures are pasted onto the scenery on the fly at the aid of image masks and warp grids. Therefore, we are able to generate a photo animation. Several examples such as ocean, pond, and ripple are included for demonstration.

KEY WORDS
Texture Synthesis, Temporal Texture Synthesis, and Warp

1. Introduction

Imaginary scenes are not necessarily hypostatized but may be rendered, such as virtual reality, computer-aided design, and image-based rendering. Texture mapping is widely used in this domain and works well in practice. Texture synthesis is an accepted way to generate textures, which can be made any proportions to fit the sceneries. However, a number of phenomena, such as a fluctuating pond, a waving lake, and a flowing river, have intrinsic motions. Static texture images may be incompetent for these appearances. A further solution to construct vivid scenes is using temporal textures.

Temporal textures are motions with indeterminate extent both in space and time [1]. They can be used to represent diverse dynamic natural phenomena such as smoke and water. Many existing algorithms attempt to produce temporal textures by mathematical simulation [2]. However, they have to develop particular formulas for specific textures. Recently, several techniques have emerged in sample-based methods which model true video clips and synthesize new video sequences [1, 3, 4, 5, 6, 7]. Our algorithm is also a kind of sample-based approach, but specially, the input sample needed for our process is only a static texture image.

We present a novel temporal texture synthesis algorithm which only need a single texture image as input to synthesize temporal textures that approximate true video clips. More specifically, our synthesized temporal textures are spatially toroidal, so that they can move in any direction toroidally. It makes our temporal textures capable of moving in any direction at will. Several examples such as ocean, pond, and ripple are included for demonstration.

Furthermore, we extend our algorithm to interactive rendering. Some variants of this practice have been considered before [6, 7, 8]. We show the power of our algorithm on activating the sea in a photo, transforming the sea to a river, or building a pond in a garden.

1.1 Related Work

There are many researches in synthesizing temporal textures and they worked well. Wei and Levoy [1] modeled the input video as 3D neighborhoods. Although accelerated by vector quantization, their pixel-wise algorithm is still time-consuming. And the motion shown in their output videos are anyway similar to the input videos. Thus, the motion direction can not be changed at will. Schödl et al. [3] proposed a term called video texture. They decomposed the input video into frames or clips and recombined them by the assists of transition probabilities. Although the video texture can produce temporally infinite output videos but it is not spatially extended. Joseph et al. [4] analyzed the whole input signals and constructed hierarchical multiresolution analysis trees, called MRA trees. They transformed the newly statistically merged MRA trees back into signals, yielding output textures. This technique is unable to create infinitely long sequence. In the meantime, their algorithm requires a large number of memory. Kwatra et al. [6] used
graph cut to synthesize 2D and 3D textures. In synthesizing temporal textures, the input videos were concatenated with optima seams, which are irregular 3D cut surfaces. They allowed for interactively merging images but did not address the problem of video-based rendering. Bhat et al. [7] captured the motions of textured particles in the input video along user-specified flow lines. This technique can warp the textures along the input flow lines and synthesize textures over the edited flow lines to build virtual landscape animations. In this paper, we also use warp technique, but we render scenes by means of user-specified image masks.

We present a novel algorithm for temporal texture synthesis. In particular, our algorithm requires only a texture image as input. Besides, our synthesized temporal textures can move in any user-specified direction at run time. This is a special problem that previous works did not address. And moreover, our algorithm allows for interactively rendering video scenes.

1.2 Overview

In section 2, we introduce the temporal texture synthesis algorithm and show some basic results. In section 3, we extend our algorithm to allow for interactively video-based rendering. Some results of landscape animations are presented in section 4. Finally, conclusion and future work are included in section 5.

2. Temporal Texture Synthesis Algorithm

Our temporal texture synthesis algorithm has the following main components: (1) texture sequence synthesis algorithm; (2) transition probabilities mapping; (3) blending rendering; and (4) controlling moving direction. In the beginning, a source image is given as the input texture, we synthesize an output sequence which is approximated to a true video clip. The second, in order to create the infinitely long video, we compute the transition probabilities by which a frame can jump to another frame without abrupt discontinuities when playing the sequence. The third, we improve the quality of the synthesized temporal textures by a cross-fading technique. Then, the user can specify the moving direction of the temporal texture using mouse drag-and-drop.

2.1 Texture Sequence Synthesis Algorithm

We now describe our texture sequence synthesis algorithm. Temporal textures are sequence of images of moving scenes that exhibit certain stationary properties in time [5]. There is a certain temporally coherent inherence in the sequence. We postulate that any two adjacent frames in temporal textures remain similar but different, in other words, some pixels occurred in previous frame may move slightly and some may dissolve into neighborhoods. In the light of this viewpoint, we are able to modify a 2D texture synthesis algorithm to synthesize texture sequences.

Figure 1 Chessboard filling texture synthesizing. (left) is the input texture. (right) denotes an output texture.

We first synthesize an output texture as the first frame of the sequence by the proposed chessboard filling texture synthesis algorithm. Figure 1 shows the illustration of the synthesis process. It is triggered from Liang’s work[9], but we present a heuristic method that is specialized from the previous work. At first, patches in the input texture are chosen randomly and pasted onto the output texture in stagger order. As shown in Figure 1(right), the initialized output texture looks like a chessboard, in which the black grids denote the patches we put at first and the white grids denote the holes. Now, the work for us to do is to fill the holes one patch per hole. Therefore, only patches in the half area of the output texture are needed to be determined.

Figure 1 also shows the illustration of the edge handling. For simplicity, we only depict the lower right edge. The edge is handled toroidally.

Due to the fixed boundary size, our algorithm is ease to accelerate. Similar to Liang [9], our chessboard filling texture synthesis algorithm is amenable to acceleration. We also speed up the patch searching process by combining Quadtree Pyramid [9], Principal Components Analysis [10], and KD-Tree [11] techniques.

Figure 2 Output frame initializing in synthesizing texture sequence. The picture in top row is the input texture. F_{i-1} and F_i denote two adjacent frames in the output sequence.

We now extend the chessboard filling texture synthesis algorithm to the texture sequence synthesis algorithm. The basic idea is as follows. For each frame in the synthesized sequence, the positions of the initial patches
obtained from input texture are recorded. Before synthesizing the subsequent frame, we disturb the positions of the initial patches. Then, the disturbed frame acts as an initialization for the subsequent frame. Figure 2 shows the illustration of this turbulence process. In the hole filling synthesis process, there are two constraints on the patch searching activities. The matched patches must yield to the similarity of spatial neighborhoods (O-shaped boundary zone), as shown in Figure 1. Meanwhile, they must yield to the similarity of temporal neighborhoods, which are the patches at the same position in the previous frame, as shown in Figure 3.

![Figure 3 Temporally matching. The candidates in the set of spatially matched patches are then extracted by the similarity with the patch at the same position in the previous frame.](image)

Let $F_i$ denotes the $i$-th frame in the synthesized sequence. Let $B_{i,k}$ denotes the $k$-th hole in $F_i$, and all the sizes of the patches to fill the hole are $(W+2E)^2*(W+2E)^2$, where the inner hole size is $W^2$ and the boundary zone size is $E$, as shown in Figure 1. While filling $B_{i,k}$ in $F_i$, we form a set $\{SPMA\}_{i,k}$ which consists of those candidate patches that satisfy the spatial constraints, as shown in (1).

$$\{SPMA\}_{i,k} = \{c | c \in T_{in}, SPADIS(c, E_{i,k}, E) \leq \xi_S\},$$  

where $c$ represents the candidate patches, $T_{in}$ is the input texture, $SPADIS(\cdot)$ is L2 distance function and denotes spatial distance, $E_{i,k}$ is the boundary zone of $B_{i,k}$, and $\xi_S$ is the spatial tolerance defined in (2).

$$\xi_S = \epsilon \left[ \frac{1}{U} \sum_{l=1}^{U} (E'_{i,k})^2 \right]^{1/2},$$  

where $U$ is the number of pixels in the O-shaped boundary zone, $E'_{i,k}$ is the $l$-th pixel value in $E_{i,k}$, and $\epsilon$ represents the spatial parameter used to control the quality of the output texture $F_i$. Referring to [9], we set $\epsilon = 0.2$. And the parameter can be controlled by the user.

After $\{SPMA\}_{i,k}$ comes into being, we form a subset $\{TIMA\}_{i,k}$ from it. $\{TIMA\}_{i,k}$ consists of those candidate patches that satisfy the temporal constraints, as follows.

$$\{TIMA\}_{i,k} = \{c' | c' \in \{SPMA\}_{i,k}, TIMDIS(c', B_{i-1,k}) \leq \xi_T\},$$  

where $c'$ represent the candidate patches. $TIMDIS(\cdot)$ is also a L2 distance function and denotes temporal distance, $\xi_T$ is the temporal tolerance, as shown in (4).

$$\xi_T = \delta \left[ \frac{1}{V} \sum_{t=1}^{V} (B'_{i-1,k})^2 \right]^{1/2},$$  

where $V$ is the number of pixels in $B_{i-1,k}$, $B'_{i-1,k}$ is the $t$-th pixel value in $B_{i-1,k}$, and $\delta$ represents the temporal parameter used to control the similarity between $B_{i-1,k}$ and $B_{i,k}$. The smaller the $\delta$, the more similar between $B_{i-1,k}$ and $B_{i,k}$. Meanwhile, we must maintain the variational characteristic of temporal textures. Empirically we set $\delta = 0.2$. Then, we select a patch randomly from $\{TIMA\}_{i,k}$ and paste it onto $B_{i,k}$. But, in the case that the number of elements in $\{SPMA\}_{i,k}$ or $\{TIMA\}_{i,k}$ is zero, we will select the nearest.

The texture sequence synthesis algorithm proceeds as follows.

1. Synthesize the first frame $F_1$ by using the chessboard filling texture synthesis algorithm. Set $i = 2$.
2. Disturb the positions where the initial patches were obtained in the input texture, and initialize the current frame $F_i$.
3. For each patch $B_{i,k}$ in $F_i$, form $\{SPMA\}_{i,k}$ and $\{TIMA\}_{i,k}$.
4. Randomly select a patch from $\{TIMA\}_{i,k}$ and paste it onto $B_{i,k}$.
5. Repeat steps 3 and 4 until finish synthesizing $F_i$.
6. Set $i = i + 1$. Repeat steps 2 ~ 5 until finish synthesizing the texture sequence.

Now we have a synthesized sequence that resembles a true video clip. Although the continuity of our sequence does not attain to the extent of the true video clip, it still be tolerable. In section 2.4, we will address this problem in detail.

![Figure 4 Frame sequence and transitions](image)

2.3 Transition Probabilities Mapping

An important state of temporal textures is “timeless” quality [3]. That is, the video sequence can be played infinitely without any visible discontinuities. Monotonic repeats look mechanical, and looping back on itself prone to appear obtrusive change. One possible solution is the video texture technique introduced by Schödl [3]. Figure 4
4 shows the illustration of this process. After playing frame $F_i$, the next frame $F_j$ is selected according to appropriate probabilities.

Our synthesized sequence, however, is different from Schödl’s. The playing order form $F_i$ to $F_{i+1}$ is not a true natural sequence. In view of this, we modify the probability function introduced by Schödl, and our function is defined as (5).

$$P_{ij} \propto \exp(-D_{ij} / \alpha D_{ij}), \quad (5)$$

where $P_{ij}$ is the probability of transition from frame $F_i$ to $F_j$. $D_{ij}$ denotes the L2 distance from frame $F_i$ to $F_j$. $\alpha$ is the probability parameter.

Sometimes, frame $F_i$ is so similar to $F_j$ that frame $F_i$ and $F_j$ are the next frame to display after each other. Since our sequences are synthesized frame by frame, there implicit causalities between every two adjacent frames. Intuitively, we like to enhance the probabilities of playing in original order. But, after playing the frames near the rear, there should be higher probabilities to loop back on to a proper frame.

Formally, let $P_{i,i+1}$ is the original probability of transition from frame $F_i$ to $F_{i+1}$. We calculate the weighted probability $P'_{i,i+1}$, as defined in (6) and (7).

$$P'_{i,i+1} = \frac{W_i \cdot P_{i,i+1}}{W_i \cdot P_{i,i+1} + \sum_{j \neq i} P_{ij}}, \quad (6)$$

$$W_i = \left( \frac{i}{n} \right)^d (1 - \omega) + \omega, \quad (7)$$

where $1 \leq i \leq n$, $n$ is the number of frames in the original synthesized sequence. $d$ is the parameter used to control the curved status of gradually decreasing of $W_i$. $\omega$ is the maximum value of $W_i$. Empirically, we set $d = 3$, $\omega = 3 \cdot n$, and these can be controlled by the user.

By using the weighted probability density function, we can create sequences with timeless and quasi-periodic qualities.

2.4 Blending Rendering

Although our texture sequence synthesis algorithm cooperating with weighted transition probabilities is able to provide temporal textures of arbitrary length with semi-regular quality. As we have mentioned before, our sequences are synthesized from a single texture image. Sometimes, there still appear noticeable discontinuities at run time. Referring to Schödl [3], we also combine our technique with cross-fading to reduce discontinuities in the temporal textures.

We use a Gaussian kernel to blend together the frames of the display sequence before and after the transition. In practice, our cross-fading function is defined as (8).

$$R_i(x, y) = \sum_{j=-\Gamma}^{\Gamma} GK(j) \ast Z_{i+j}(x, y), \quad (8)$$

where $R_i$ represents the display frame after blending. $Z_i$ represents the current frame. $GK()$ is a Gaussian kernel.

After using cross-fading, the discontinuous effects are reduced. However, the display sequences maintain a certain level of blur. But, cross-fading improves the quality of our temporal textures effectively.

![Figure 5 A visual window moving in a picture](image)

2.5 Controlling Moving Direction

The user can easily specify the moving direction and speed of our synthesized temporal textures. Because all the frames in our sequence are toroidal, the display frames after processed by cross-fading are still toroidal. Therefore, the display frames can be moved toroidally and create no visible seam. Figure 5 illustrates this process. When the window in Figure 5 moves to the upper right, the scenery in the window looks moving to the lower left. The moving direction can be specified by just mouse click. And the moving distance in each step also can be input by the user by means of a dialog. Hence, our synthesized temporal textures are able to move in any direction and be controllable.

![Figure 6 Results for temporal texture synthesis. The top row shows the input texture images, while the bottom row shows the synthesized temporal textures](image)
Table 1 Output resolutions and running times in seconds

<table>
<thead>
<tr>
<th>Input Texture</th>
<th>Sequence Synthesis (20 frames)</th>
<th>Calculating Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocean</td>
<td>67s</td>
<td>10s</td>
</tr>
<tr>
<td>pond</td>
<td>81s</td>
<td>10s</td>
</tr>
<tr>
<td>ripple</td>
<td>61s</td>
<td>10s</td>
</tr>
</tbody>
</table>

2.6 Basic Results

We applied our algorithm so far to several textures, such as ocean, pond, and ripple. Our system environment is as follows: Intel Celeron 2G Hz with 256 MB RAM, Ms Visual C++ 6.0, and OpenGL. Figure 6 shows some of our results for temporal texture synthesis. The resolutions of the synthesized temporal textures are all 256*256. The patch sizes are all 16*16 and the boundary zone sizes are all 8. Once the basic texture sequence is synthesized, the rendering process including cross-fading is real-time. Table 1 summarizes the running times for those results in Figure 6. It not only runs much faster then previous techniques but also requires only a texture image as input to synthesize temporal textures with timeless, spatially extensible, and quasi-periodic qualities.

\[
AR(x, y) = MA(x, y) \ast WB(x, y) + (1 - MA(x, y)) \ast LA(x, y),
\]

where AR denotes the artificial landscape, as shown in Figure 7(d). MA denotes the mask image. WB denotes the mapped warp band. LA denotes the original landscape photo. At run time, our system can move and map the display frame on the fly. Therefore, our temporal textures can be controlled to move along the warp band.

3. Extensions

Our synthesized temporal textures can be extended to create artificial landscape animations. Figure 7 shows an example. The user can design the mask at will by mouse drag and drop, as shown in Figure 7(a). Our system will make the area closed automatically and fill the area by blood-fill algorithm. And, we still can use a average filter to blend the mask edges. After the mask image is finished, the user is able to design the warp grids by mouse drag, as shown in Figure 7(b). There are three types of warp grids available, rectangle, clockwise sector, and counter-clockwise sector. There are unlimited combinations of these grids. After the warp grids are achieved, our algorithm automatically picks the patches from the temporal texture toroidally and maps them onto the warp grids respectively, as shown in Figure 7(c). Finally, the artificial landscape is rendered as follows.

Figure 7 In (a), the user design the mask image. In (b), the user designs the warp grids. In (c), the temporal texture is mapped onto the warp grids. (d) is the composite landscape animation.

3. Extensions

Our synthesized temporal textures can be extended to create artificial landscape animations. Figure 7 shows an example. The user can design the mask at will by mouse drag and drop, as shown in Figure 7(a). Our system will make the area closed automatically and fill the area by blood-fill algorithm. And, we still can use a average filter to blend the mask edges. After the mask image is finished, the user is able to design the warp grids by mouse drag, as shown in Figure 7(b). There are three types of warp grids available, rectangle, clockwise sector, and counter-clockwise sector. There are unlimited combinations of these grids. After the warp grids are achieved, our algorithm automatically picks the patches from the temporal texture toroidally and maps them onto the warp grids respectively, as shown in Figure 7(c). Finally, the artificial landscape is rendered as follows.

AR(x, y) = MA(x, y) * WB(x, y) + (1 – MA(x, y)) * LA(x, y),

where AR denotes the artificial landscape, as shown in Figure 7(d). MA denotes the mask image. WB denotes the mapped warp band. LA denotes the original landscape photo. At run time, our system can move and map the display frame on the fly. Therefore, our temporal textures can be controlled to move along the warp band.

4. Results

Figure 8 shows several examples that the static photos of real scenes are reconstructed using our synthesized temporal textures. The left column shows the original photos and the right column shows the artificial landscape animations. The results look like true videos. The top row shows ocean along a cliff. We directly take the ocean in the original photo as input texture to synthesize the temporal texture, which is used to make the ocean current moving. In the next row, we transform the ocean to a flowing river by using the temporal texture shown in Figure 7(left). The last row shows a garden. We always wish to have a garden with a pond. Now we make the imagination visible by using the temporal texture shown in Figure 7(middle).

Figure 8 The examples of artificial landscape animations
5. Conclusion

Temporal textures are interesting and important for a wide variety of animations in computer graphics. We have presented a novel algorithm for temporal texture synthesis. Specially, our synthesized temporal textures can move in any user-specified direction at run time. We demonstrate several examples such as ocean, pond, and ripple. Our algorithm can be extended to allow for interactively rendering. We also show an ocean current along a cliff and a waving pond in a garden. So in other word, we animate a static photo.

We have successfully synthesized temporal textures whose moving direction is controllable, but there still are some limits we want to break through. Because our temporal textures are synthesized from a texture image, we may use cross-fading technique to mitigate the discontinuities. Therefore, the sequences will maintain a certain level of blur.

At the future work, we are interesting in the temporal distance between two frames. Most previous works use L2 distance to represent the difference between two images [8]. L2 distance is able to compute the total difference value of the corresponding pixels, but it is unable to compute the correlation. This problem can be addressed by researching the similarity between two images or the correlation between two adjacent frames in a video. By solving this problem we can make the temporal texture synthesis results more similar to true videos.

Acknowledgement

We would like to thank the anonymous reviewers for their helpful comments regarding earlier drafts of this manuscript.

References


