Columbia Computer Science Technical Report (2005)

Time-Varying Textures

Sebastian Enrique [1], Melissa Koudelka [2], Peter Belhumeur [1], Julie Dorsey [2], Shree Nayar [1] and Ravi Ramamoorthi [1]

[1] Columbia University [2] Yale University

Abstract

Essentially all computer graphics rendering assumes that the reflectance and texture of surfaces is a static phenomenon. Yet, there is an abundance of materials in nature whose appearance varies dramatically with time, such as cracking paint, growing grass, or ripening banana skins. In this paper, we take a significant step towards addressing this problem, investigating a new class of time-varying textures. We make three contributions. First, we describe the carefully controlled acquisition of datasets of a variety of natural processes including the growth of grass, the accumulation of snow, and the oxidation of copper. Second, we show how to adapt quilting-based methods to time-varying texture synthesis, addressing the important challenges of maintaining temporal coherence, efficient synthesis on large time-varying datasets, and reducing visual artifacts specific to time-varying textures. Finally, we show how simple procedural techniques can be used to control the evolution of the results, such as allowing for a faster growth of grass in well lit (as opposed to shadowed) areas.

1. Introduction

A great deal of work in texture analysis and synthesis has been presented in both computer graphics and computer vision in recent years. However, texture has usually been considered static—the surface itself remains constant through time. Many real-world textures are of interest, however, expressly because of the way their appearance changes or evolves with time. Consider the fracture of mud drying in a riverbed, the growth of grass on a hillside, or the formation of oxides on copper. Each of these natural processes forms a pattern over time, often producing striking effects.

The modeling of these processes, and more generally pattern formation in nature is a difficult problem, that has long been studied in biology, physics and mathematics (see [Mei92, CH93, Ada03] for reviews). While some significant progress has been made, it is generally recognized [Bal99] that the real world is far too complex to be described by a limited set of mathematical tools.

In this paper, we avoid the inherent difficulties in mathematical modeling by developing an example-based approach. We use time-lapse images to capture the complexity in a range of real world processes, including biological growth and decay, breakdowns such as stress fracture, accumulation of particles, and state changes such as oxidation. We then synthesize larger spatial patterns of these processes using a new time-varying texture synthesis algorithm (for example, see figure 1). Our specific contributions are:

Database of time-varying textures: This paper presents an initial dataset of time-varying textures (some of which are shown in figure 2), which can be a useful resource for future efforts. Capturing the time variation of appearance is a difficult problem, since many phenomena such as grass growth occur over fairly long timescales, making acquisition challenging. Furthermore, to be usable for texture synthesis, the acquisition must be carried out in a carefully controlled manner, preserving parameters such as lighting and humidity over the entire duration. This is an ongoing effort, and the entire database, which is the first such data on time-varying appearance, will made available upon publication.

Time-varying texture synthesis algorithm: Synthesizing textures that vary across time is a challenging problem because the appearance can evolve and change completely. In this paper, we take an important first step, showing how to adapt quilting-based methods for static textures to synthesize time-varying textures. We do so by addressing three important challenges. First, a synthesis technique must preserve the temporal coherence or continuity in the original data—we cannot simply synthesize each frame separately. Second, dealing with large time-varying datasets requires significant computation, making efficiency a concern. Finally, we must address temporal visual artifacts (such as cracks in paint appearing at the same time in various parts of the synthesized texture if we paste the same block from the input image in various locations). We solve these problems to develop a simple synthesis algorithm that works well for a wide class of time-varying textures.

Controllable time-varying texture synthesis: In many real-world situations, the time evolution will be different for different locations in the texture. For instance, grass will grow faster in well lit areas rather than those in shadow. The rate at which paint cracks may depend on the curvature of the surface. These are issues that do not arise for static textures and have therefore not been addressed in previous work. We develop a set of simple procedural tools that allows a user to control the rate of time-variation of different locations on the texture, allowing for controllable time-varying texture synthesis.

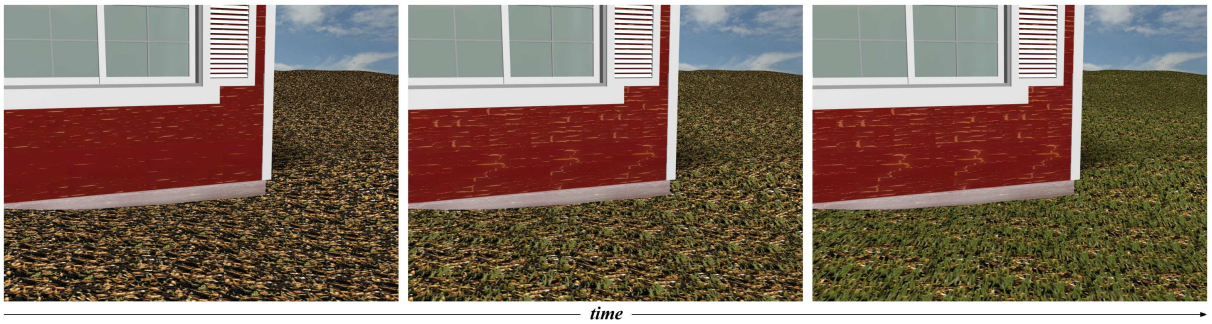

Figure 1: *Sample images of a time sequence with synthesized time-varying textures for cracking paint and growing grass (the input sequences are found in rows 1 and 2 of figure 2). This paper offers a new capability in computer graphics, of rendering these changes in appearance over time, that cannot be addressed by current methods for static textures.*

2. Previous Work

Our work is related to much of the recent literature in texture synthesis, motion or dynamic textures, and the study of pattern formation and weathering in computer graphics.

Texture Synthesis: There is a great deal of recent work on synthesizing texture by example. One approach is to use *parametric* [HB95, PS00] models; these are generally more useful in characterizing textures for analysis, rather than synthesis. Another approach is to use *non-parametric* methods, with a collection of examples from the source image used to represent the texture. DeBonet [DeB97] generates textures by sampling from multiscale filter responses. In [EL99], pixels are copied directly from the sample texture. [WL00] extended this work, using a multiscale approach and vector quantization for faster processing. Their algorithm was extended to synthesis on 3D surfaces in [WL01] and [Tur01], and to synthesis on 3D surfaces with lighting and viewpoint variation in [TZL*02]. More recently, several algorithms have synthesized textures by placing *entire patches* from the sample image [EF01, LLX*01, KSE*03]. Patch based methods have generally demonstrated the best and most efficient sample-based texture synthesis results to date.

All of these methods have considered static textures. As discussed in the introduction, the extension to time variation is challenging, since we must consider temporal coherence, efficiency and reduction of perceptually distracting temporal artifacts. One of the main contributions of this paper is to solve these problems in order to adapt patch-based texture synthesis algorithms to synthesize time-varying textures.

Motion texture: Work termed as dynamic texture has been studied in tcomputer graphics and computer vision [SP96, SSSE00, SDW01, WZ02]. However there are some important distinctions between that work and our own. First, the "texture" in previous work has often been a result of repetition in time, arising from motion in the sequence (e.g., fountains or waterfalls within a larger scene, ripples on the surface of water). In contrast, we look at surfaces that meet the traditional definition of texture in each frame (i.e., the texture is stochastically stationary and repeats *in space*—but the visual appearance of the texture is changing over time). Previous work with time-varying texture is perhaps more aptly referred to as "motion texture." Furthermore, time repetition is key to the operation of these dynamic texture algorithms, since they are based on motion models or similarity between frames. Our datasets are by definition changing their appearance completely over time, so there is no notion of repetition. For this reason, the methods used in previous work do not apply to the datasets we consider.

Appearance	Time-Varying Appearance
BRDF	TBRDF
TF (Texture Function)	**TTF (Time-Varying Texture Function)**
BTF [DVGNK99]	TBTF

Table 1: *Extension of common appearance concepts such as the BRDF, textures and light and view varying textures (BTFs) to time-varying appearance. In this paper, we focus on the time-varying texture function (TTF).*

Pattern Formation: Some recent work has focused on physical simulation of the patterns generated by weathering ([DEWJ*99] and cited references) and corrosion [MDG01]. Earlier work has focused on reaction-diffusion methods [Tur91, WK91] and L-systems for plant geometry [PL90]. A common thread in these methods is that the textures are procedurally generated, through mathematical equations and simulation. As with any procedural texture, synthesis is often difficult to control, and achieving a result that matches a given sample requires a great deal of trial and error. Our method has the advantage that it is derived from the sample itself, and so generating a result matching a desired appearance relies only on obtaining a sample.

3. Time-Varying Appearance

We first formalize the notion of time-varying appearance. One can imagine extending many common appearance concepts, such as the BRDF, textures, and the Bi-directional Texture Function (BTF [DVGNK99]) for light and view variation, to include an additional time dimension as shown in table 1. In this paper, we focus on textures, introducing the Time-varying Texture Function or TTF, assuming lighting and viewpoint are fixed. The extension to TBRDFs and TBTFs remains a rich area of future work.

The TTF concept is a simple extension of static texture, and we can denote the time-varying texture p as

$$\text{TTF} = p(x, y, t), \tag{1}$$

where (x, y) is the spatial location on the surface (or equivalently, the corresponding pixel in the image) and t is the time (which in previous work has been assumed static).

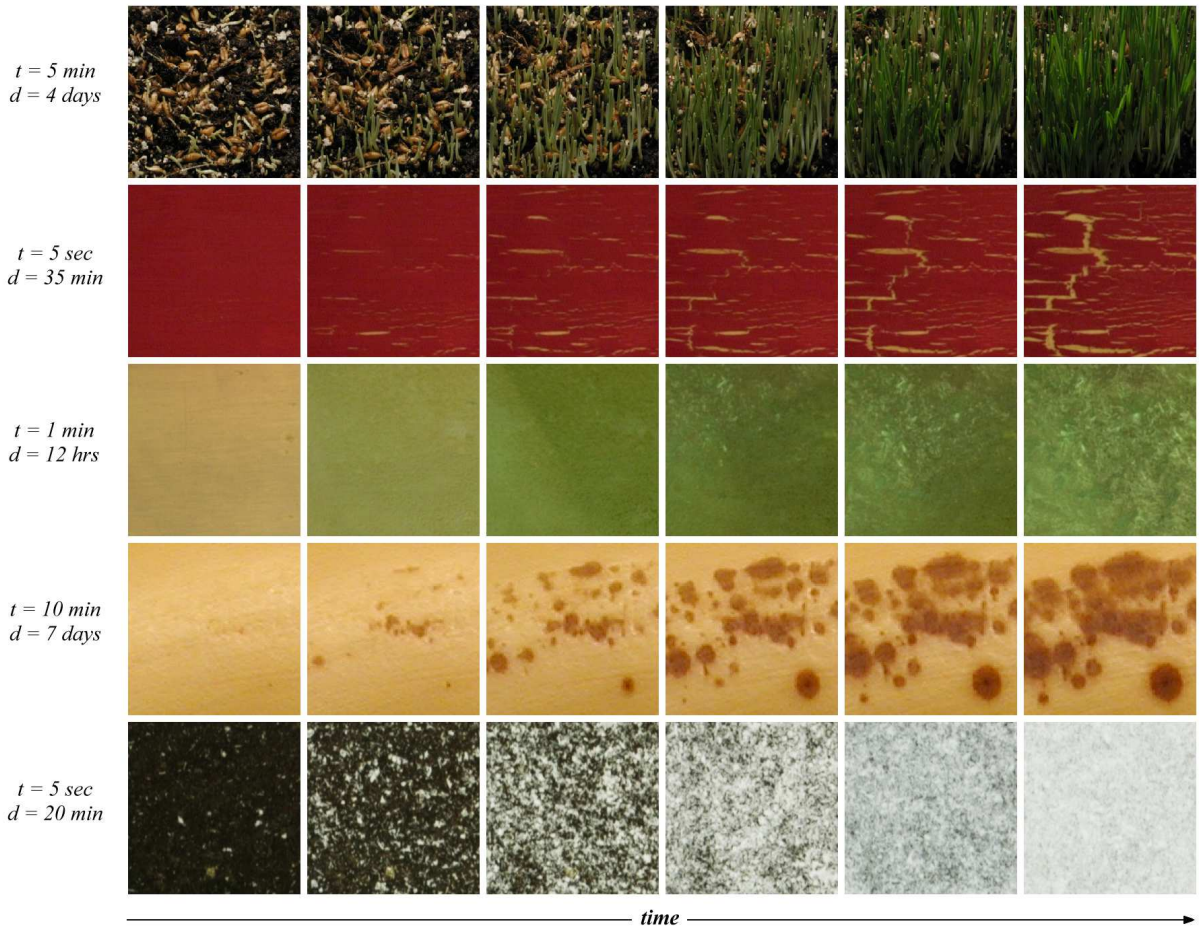

Figure 2: *Examples from our database of time-varying textures. From top to bottom, grass gowth, cracking paint, copper patination, ripening banana, and snow accumulation. Images were acquired with controlled lighting conditions at times t apart over a total duration d. Each dataset contains from 200 to 2500 images taken at intervals ranging from $t = 5$ sec to $t = 10$ min, and over a duration ranging from $d = 20$ min to $d = 7$ days. This time-varying appearance database is the first of its kind and represents a valuable resource for future research efforts.*

4. TTF Acquisition

The first step in investigating time-varying textures is to acquire datasets representing them—some examples are shown in figure 2. Acquiring good datasets is a challenging problem for several reasons. First, many of the natural processes we consider occur over a considerable duration (several days for grass growth and ripening of banana skin). Second, it is difficult to control conditions such as the lighting over this duration, which is essential to obtain good data for texture synthesis. Given the considerable difficulty in acquiring calibrated time-varying data, we believe this database to be an important contribution; we will extend it to more natural processes and make it available upon publication.

Images of our acquisition for several of the textures is shown in figure 3. We acquire images using time-lapse photography, recording photographs at given time intervals. These time intervals ranged from five seconds (paint drying and cracking, snow accumulating on slate) to between 1 and 10 minutes (grass growth, oxidating copper, ripening banana), and the durations over which the images were captured ranged from twenty minutes (snow) to several days (grass and banana). In all, we obtained between 200 and 2500 images for each time-varying process.

Since the definition for TTFs has the lighting and viewing angles fixed, the acquisition system must have controlled viewing conditions over time. We used a fixed camera and fixed incandescent light sources, while the camera shutter was controlled automatically. Two separate configurations were used, depending on the time scale of variation. For processes that happen over a longer period of time (e.g. several days, as with grass growth), a Nikon CoolPix 990 digital camera was used with camera control via the USB port on a personal computer. The shutter could be triggered up to every 30 seconds, with an accuracy of plus or minus a few seconds. For faster processes that happen over a few minutes up to a few hours (e.g snowfall), a Nikon D1 digital camera was used with camera control via a PocketWizard MultiMax transceiver remote triggering device. The shutter could be triggered as fast as every second with very high accuracy.

The illumination must also be carefully controlled to maintain consistency across the entire dataset. For processes that happen on a short time scale (e.g., snow falling, where only natural light is used), the lighting is assumed to be approximately constant. For longer acquisitions (e.g., grass growth), however, we controlled the lighting by enclosing the sample and camera in a light-tight aluminum box with

Figure 3: *Images of acquisition for (from left to right) growing grass, drying and cracking paint, patinating copper, and ripening banana.*

one or more fixed incandescent bulbs (for most acquisitions, two 60W bulbs) for illumination.

Many of the samples also required a somewhat specialized set-up to achieve the desired time-varying texture effect. In some cases, this is straightforward. For instance, for the snow dataset, we simply positioned a camera above a slate tile during a snowstorm. The slate was cleared of all snow before acquisition, and images were acquired every few seconds as the snow accumulated. For the ripening banana dataset, we used several bananas placed in a light-tight aluminium box and allowed to ripen over approximately one week. Two fixed incandescent bulbs were the only light sources in the box. Below, we discuss in somewhat more detail, the setup for the first three examples in figure 2.

Grass Growth: Achieving grass growth proved difficult in the enclosed environment of the light tight box. Therefore, we used Cosmic Kittycat grass, a fast growing grass commonly available at pet stores, that is packaged in its own soil. After a three day germination period, where the seeds were kept in a dark, moist location, the grass began to grow. It was then placed in the light tight box, and the grass grew rather quickly. The box lid was left slightly open and covered with blackout fabric for ventilation. Two fixed incandescent bulbs provided the only lighting.

Weathered Paint: We used a pine board with DecoArt weathered wood cracking medium and red acrylic paint. The weathering medium was applied to the board and allowed to dry for 20 to 30 minutes. The paint was then applied and the dataset acquired in open air with two fixed incandescent light sources. The size of cracks could be controlled by changing the thickness of the weathering medium or the paint.

Copper Oxidation: We used a piece of copper sheeting with Sophisticated Finishes Patina Green Antiquing Solution. The sample was acquired in open air to speed evaporation time, and blackout fabric was used to keep the lighting, provided by two fixed incandescent bulbs, consistent throughout the acquisition. The liquid patination solution was applied to the copper and oxidation visibly began immediately. As the chemical evaporated, the patina formed.

Note that all of our acquisitions are assumed independent of geometry, and are acquired on planar surfaces or as close to planar as possible in the case of the banana, for example. Although non-planarity and geometric deformations do lead to movement of surface points in the image, we consider this effect to be relatively small—however, it may lead to some degradation in the visual quality of our final results.

5. TTF Synthesis

Many algorithms for synthesizing textures in both 2D and 3D have been introduced in recent years, as discussed in Section 2. In general, the most successful algorithms have relied on copying pixels or patches directly from the sample image and finding best fit boundaries between them. It would therefore be ideal to find a simple adaptation of these approaches to synthesize time-varying textures. This would also enable the many recent extensions and improvements to texture synthesis to also apply immediately to TTFs.

However, one cannot directly apply previous methods to TTF synthesis. The problem is difficult because the texture can evolve, completely changing its appearance over time. There are three main challenges we address in this work. First, we must preserve the temporal coherence and gradual time variation in the original input sequence—something which static texture synthesis methods do not address. Second, we must address efficiency concerns given that we may be dealing with hundreds to thousands of times more data than for static texture synthesis. Third, we must alleviate visual artifacts from temporal and spatial repetitions that can arise with time-varying texture synthesis, but are not present for static algorithms. In this section, we develop a simple algorithm that solves these problems, and enables image quilting [EF01] to be adapted to time-varying textures.

5.1. Preserving Temporal Coherence

Perhaps the most basic approach is to quilt each frame independently. As shown in figure 4b, while each individual frame looks reasonable, their time sequence does not exhibit any kind of temporal coherence, leading to considerable randomness and flickering as shown in the video.

There are several possible ways to address this for time-varying textures. Instead of standard quilting, which quilts blocks that have a small spatial extent, but are fixed in time, we could consider quilting regions that are one pixel wide in space, but vary over time. In general, one might imagine quilting arbitrary spatiotemporal regions. In practice, we have found it very important to preserve the entire time variation of the original input block to maintain temporal coherence. Hence, we choose to quilt blocks that have a small spatial extent as in the original algorithm, but *extend over the entire time sequence* (figure 4c). We have found this simple approach suitable for a wide variety of time-varying textures.

Consider the input time-varying texture $p(x,y,t)$. The inner loop of the quilting algorithm involves a comparison phase, which determines which block should be chosen to be pasted in the current location. For static textures, we can form an error metric as

$$E = \sum_{x,y} (p_1(x,y) - p_2(x,y))^2, \tag{2}$$

where we are comparing small blocks p_1 and p_2. To extend

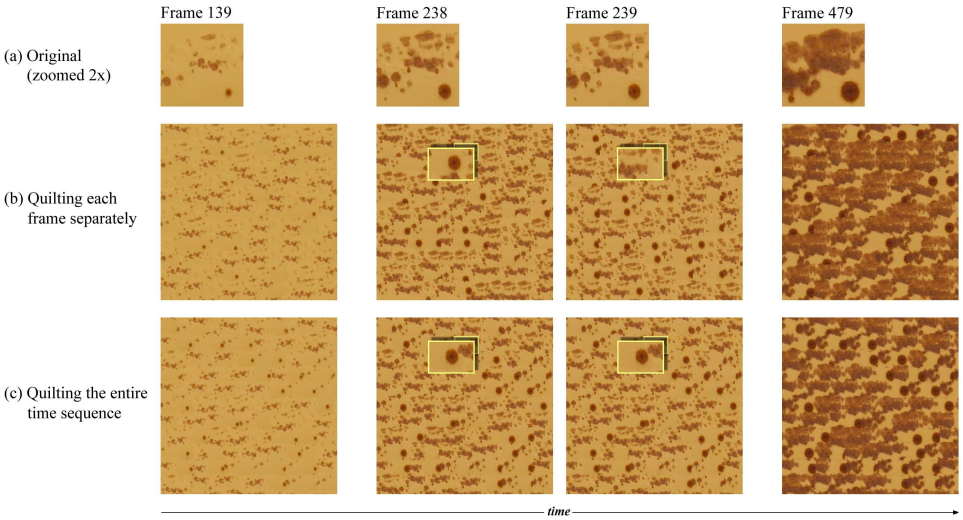

Figure 4: *Preserving temporal coherence in TTF synthesis. In (a) we show the original input data (zoomed in by a factor of 2). In (b), we synthesize each frame separately using standard image quilting, which loses temporal coherence (note the large difference in closeups for adjacent frames). In (c), we quilt over the entire time sequence, comparing and copying the full time sequence for source blocks. This approach preserves temporal coherence much better.*

this idea to the entire time sequence, we instead use

$$E_{TTF} = \sum_t \sum_{x,y} (p_1(x,y,t) - p_2(x,y,t))^2. \quad (3)$$

When we paste a block into the synthesized texture, we paste the same spatial block for the entire time sequence, preserving the original time variation. In our approach, we also use the same seam to join blocks across the entire time sequence (thus also ensuring temporal coherence of these seams), with the best seam or cut location chosen based on consideration of all the frames. In effect, instead of quilting small square blocks with spatial but no temporal extent, we quilt cuboids with a small spatial extent as before, but whose temporal extent is the entire original sequence.

5.2. Efficient Time-Varying Synthesis

The inner loop comparison in equation 3 is linear in the temporal extent (number of frames in the time sequence). Thus, if we have acquired one thousand frames, TTF synthesis will be one thousand times slower than synthesizing a single static texture. Furthermore, the memory requirements will also grow linearly with the number of frames. In this subsection, we develop an optimization using singular-value decomposition to compress the original dataset and speed up the comparisons. This approach is inspired by recent work on optimizing static 2D texture synthesis [ZG04], and techniques to compress lighting and viewpoint variation for bidirectional texture functions, such as 3D textons [LM01]. We use a linear SVD method applied to the very different problem of representing variation with time, to speed up the comparison phase of the algorithm. It should be emphasized that this compression only optimizes the comparisons in equation 3. The final output always copies blocks directly from the original input without compression.

Since textures are by definition repetitive in space, we believe that there exists a low dimensional basis for time-varying pixels. To compute this basis, we collect the time-varying texture data $p(x,y,t)$ for all m original pixel locations (x,y) and all n frames (labeled with time t) into a large $m \times n$ matrix (a typical size would be 65536×500),

$$T_{m\times n} = \begin{pmatrix} p(x_1,y_1,t_1) & p(x_1,y_1,t_2) & \dots & p(x_1,y_1,t_n) \\ p(x_2,y_2,t_1) & p(x_2,y_2,t_2) & \dots & p(x_2,y_2,t_n) \\ \dots & \dots & \dots & \dots \\ p(x_m,y_m,t_1) & p(x_m,y_m,t_2) & \dots & p(x_m,y_m,t_n) \end{pmatrix} \quad (4)$$

The singular value decomposition of T is then performed,

$$T_{m\times n} \approx U_{m\times k} \Sigma_{k\times k} V^t_{n\times k}, \quad (5)$$

where the columns of U and V hold the first k eigenvectors or singular vectors spatially and along the time axes respectively. This decomposition can be done separately for each color channel for RGB images, or we can simply consider using $3m$ pixel values. Comparison of pixels in intensity space in equation 3 for each pixel is simply replaced by comparison in time-varying pixel coefficient space, i.e. comparing the corresponding rows of U, after weighting each column by the magnitude of the singular values in Σ. The major benefit is that each row of U has only k entries, instead of the n frames in the original input.

Figure 5 (left) shows a plot of the reconstruction error using a certain number of eigenvalues. For most of the textures, a rank 15 approximation has an error of under 1%, and we obtain a speedup by one to two orders of magnitude relative to using the full time sequence. For the grass dataset, the error falls off more slowly. This is because there is considerable visual change over the length of the sequence, with the grass growing and elongating. Nevertheless, 15 eigenvalues still are accurate with only a 5% error. We also show the reconstruction and error using a different number of singular values in figure 5 (right) for the snow dataset, again demonstrating that a low rank approximation is accurate, and suitable for the comparisons in texture synthesis.

In terms of a computational and memory savings, the complexity is now linear in the number of eigenvectors k, instead of the number of frames in the original input sequence. Since we use 15 eigenvectors instead of the hun-

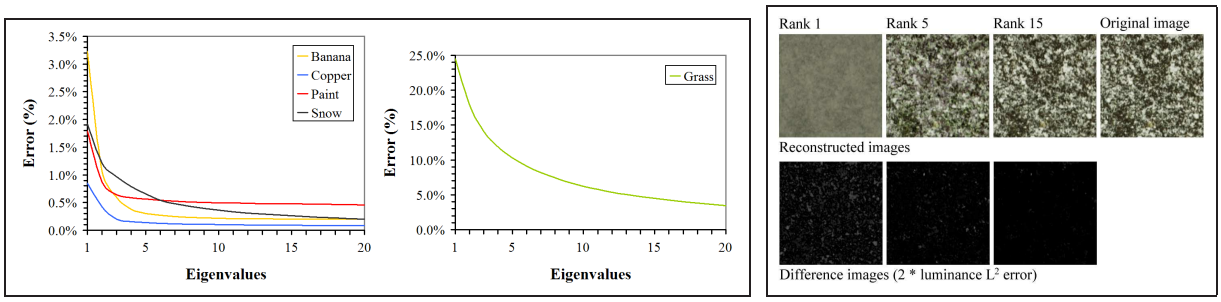

Figure 5: *Speedups of one to two orders of magnitude can be obtained by compressing the original dataset using SVD.* Left: *Graphs showing percentage RMS error in representing the various time-varying textures using a certain number of eigenvalues.* Right: *Comparison of reconstructions with different ranks on the snow accumulation dataset. The magnified difference images are shown in the bottom row. While the visual error in using a rank 1 approximation is clear, the approximation rapidly converges to the true image, with little perceptible difference in a rank 15 approximation.*

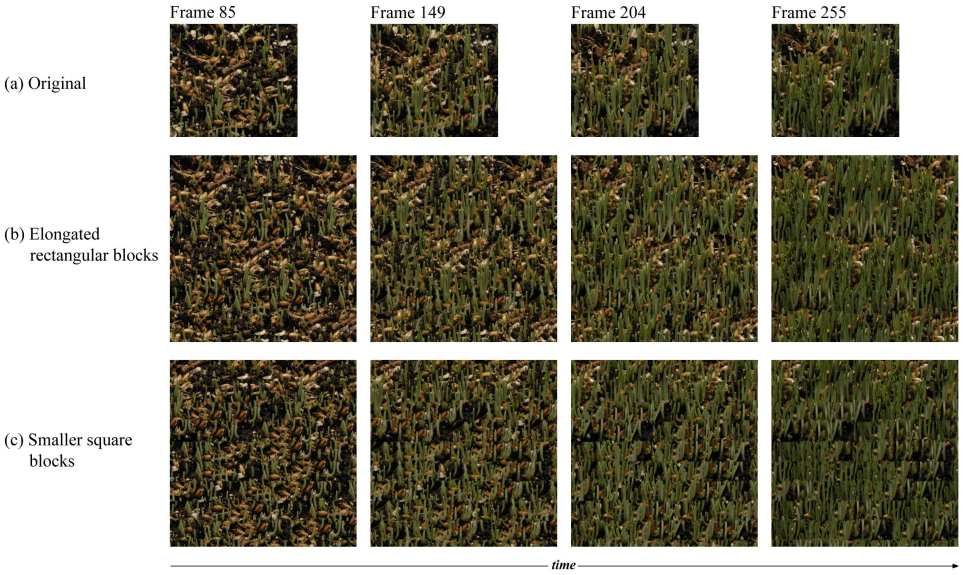

Figure 6: *Time-varying effects such as growing and elongating of grass can be addressed by setting an elongated rectangular block size for texture synthesis. (a) is the original data, (b) is using elongated* 320×64 *rectangular blocks, and (c) is using smaller* 64×64 *standard square blocks. Because the grass grows upward, the elongated blocks perform better, particularly for later frames (204 and especially 255).*

dreds to thousands of original frames, we obtain a speedup of one to two orders of magnitude. Note that we use the SVD only for comparisons, to increase efficiency, but copy the actual image data from the original time-varying texture, so there is no loss in sharpness or fidelity of the output.

5.3. Reducing Visual Artifacts

So far, we have devloped an efficient algorithm for time-varying texture synthesis that preserves temporal coherence. Good results can be obtained for most examples using this approach (see figure 4 for instance). However, in some cases, the nature of time-varying textures leads to specific visual artifacts not usually found in static texture synthesis. In this subsection, we will describe some of these issues, showing how some simple approaches to setting parameters and optimizing the algorithm can improve visual quality.

Setting block size—Elongated rectangular blocks: In standard quilting-based methods, one uses small square blocks to paste as patches, with the block size set based on the feature size of the texture. For time-varying textures, this feature size can change over time, such as when grass grows, with the blades elongating. An example is shown in figure 6. If we use 64×64 square blocks, as in figure 6c, good results are obtained for early frames (85 and 149). However, for later frames (204 and especially 255), as the grass grows longer, the tiling because of the blocks is fairly obvious. On the other hand, simply using elongated rectangular blocks of size 320×64, as in figure 6b, allows us to synthesize an effective texture over a much longer sequence.

Reducing repetitions—Jittering time sequence: In the basic algorithm, the entire time sequence of each patch is copied from the original data. Since the same patch will be placed at various parts of the image, any event in that patch (such as a crack appearing) will happen simultaneously throughout the image, which can be distracting and repetitious. Note that this is a time-related repetition (the same event happens in different places at the same time), that is not an issue for static texture synthesis.

We address this problem by creating some randomness in the time sequence for different instances of a patch in the synthesized texture. We do this by jittering the way the patch

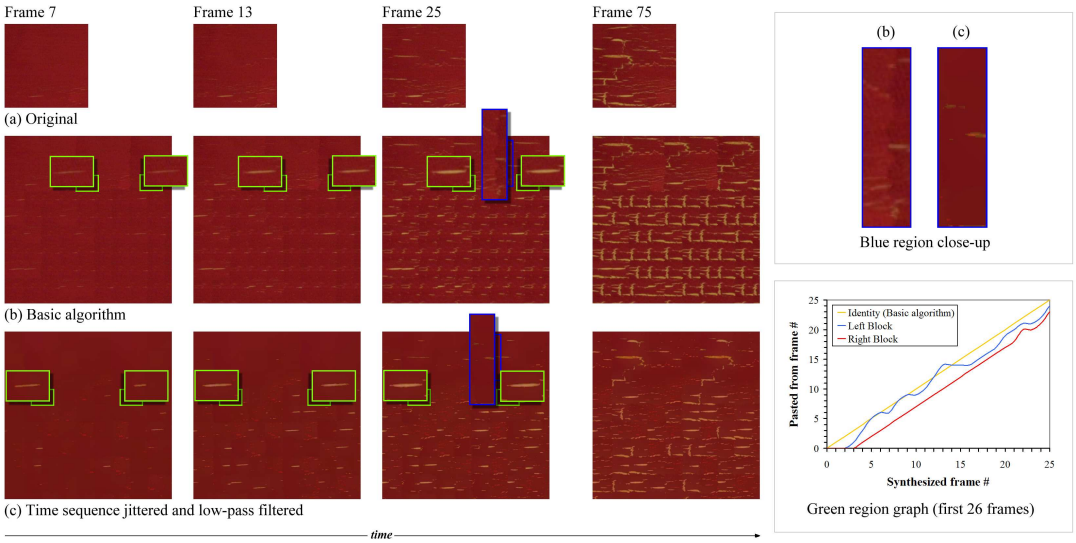

Figure 7: *The effects of some repetitions in time-varying textures can be reduced by using jittering and filtering low-frequency gradients. The top row (a) shows some early frames and a later frame from the original cracking paint sequence. Below that (b), we show the basic algorithm. In this case, there are two issues, causing the final repeated texture pattern on the right. First, the time sequence is the same for the same original block pasted at different locations in the texture (green closeups and graph at the bottom right). Second, the original paint has a low-frequency spatial gradient with a slight color shift from left to right, that restricts the choice of blocks for matching and the quality of the results (blue closeup). The lower row (c) shows a modified algorithm where the time sequence is jittered (see the green closeups now and the graph on the right) and a low-pass filter is applied before texture synthesis to better match the colors (blue closeup). The synthesized time-varying texture now more faithfully represents the original, without the repetitions observable in the basic algorithm.*

is copied from the original image. That is, instead of frame 100 coming exactly from frame 100 of the original, it may come from frame 97 or 103. To implement this,

$$p^{\text{synth}}(x,y,t) = p^{\text{orig}}(x',y',f(t)), \tag{6}$$

where the block from (x',y') in the original is pasted into the synthesized texture as usual, but we also copy the sample from a new time location $t' = f(t)$.

The interesting issue is how to construct the function f to allow jittering. Note that there are two properties we want to maintain. First, the time variation $f(t)$ should be monotonically non-decreasing. Second, it should not deviate too much from the original timeline, i.e. $\| f(t) - t \| \leq \varepsilon$ for some user-specified value of ε. A function with these properties can be constructed using a simple procedural technique with a random-number generator δ (that gives values in the range $[-1,1]$). We initially define $f(0) = 0$. Then, we simply compute for the next frame in the sequence,

$$f(t+1) = f(t) + 1 + \delta, \tag{7}$$

provided this keeps the function within the tolerance. If this would cause the tolerance to be exceeded, we set $\delta = 0$. This approach provides a simple way of randomizing the time sequence, as seen in the graph at the bottom right of figure 7.

Filtering low-frequency gradients: In certain cases, the original material has a low-frequency gradient, such as from left to right in the paint example. This makes texture synthesis hard, since patches from the right of the original sample will not fit well with patches from the left, due to the intensity difference, even if their time variations match. This can lead to two problems, as seen in figure 7 (row b) and the blue closeup. First, there can be distracting color gradients within a region of the synthesized texture. Second, because of the difficulty in finding good matches, the synthesized texture may exhibit repetitions.

To address this problem, we may low-pass filter the images (in practice, we choose a representative frame for this purpose), to obtain the low frequency gradient. The image can then be normalized with respect to this gradient, and all texture synthesis operations can be peformed on the normalized image sequence. If desired, the final result for each frame can then be multiplied by the low-frequency variation to capture the original spatial gradients.

The effects of jittering and low-pass filtering are shown to improve the quality of the synthesized paint sequence in figure 7. The basic algorithm (b) exhibits similar events happening throughout the texture due to identical patches being copied (green closeups). The color gradients also lead to a difficulty in selecting patches for texture synthesis (also leading to the repetitions in the result on the right), and create somewhat distracting small color gradients in the results. Applying jittering and low-pass filtering largely eliminates these visual artifacts in row (c).

5.4. Results

We have used our algorithm to synthesize time-varying textures for several examples in our database, including all those shown in figure 2. For example, the grass and paint results were shown in figures 6 and 7 respectively. The banana skin example was shown earlier in figure 4. Results on the snow dataset are shown later in figure 9. In all of these cases, we have been able to generate high quality time-varying textures. Since these datasets represent a reasonably wide range of natural processes, we believe that our TTF synthesis algorithm is a simple approach that can capture a fairly broad class of time-varying appearances for computer graphics.

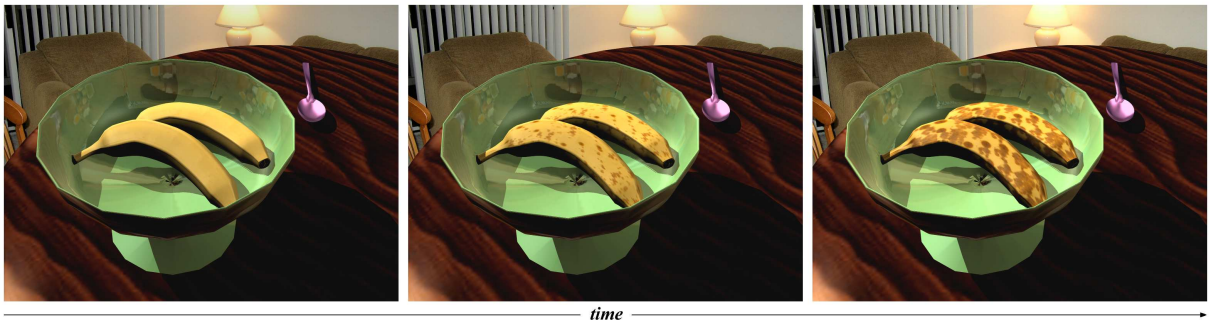

Figure 8: *A fruit bowl with ripening bananas. The banana skin is a time-varying texture, synthesized using our algorithm. Note also that we can combine dynamic TTFs with standard static textures for the table top and bowl.*

Our algorithm can also be applied with standard texture-mapping to entire scenes. In addition, TTFs can be combined with standard static textures. Figure 1 shows applying the paint TTF to the wall of the house, and the growing grass TTF to the outside, to give the time-varying appearance of paint cracking on the wall and grass growing in the garden. Figure 8 applies the banana TTF to a scene of fruit in a bowl, showing a realistic example of ripening. These scenes represent a new capability in computer graphics, of easily creating images and videos of time-varying scene appearance.

6. Controllable TTF Synthesis

In real-world textures and processes, the rate at which the process unfolds is a function of many parameters including the underlying shape, and environmental factors in the scene such as lighting, temperature, and humidity [ML93, Sti02]. For instance, the rate of grass growth will depend on the lighting (growth will be faster in well lit rather than shadowed regions). Paint may crack faster on curved surfaces. These issues are usually not relevant to static textures, and have therefore been unexplored in previous work on texture synthesis. Furthermore, it is not clear how to obtain these effects, since our input consists of only a single time-varying texture sample, and we usually do not know the exact (or even approximate) dependence on environmental parameters and spatial locations. In this paper, we address this important problem by taking a phenomenological approach, developing a set of procedural tools that are easy to specify and can be applied to obtain desirable user-controlled effects.

To control the time progression of a TTF, we resample the data based on some property of the surface or environment. The desired resample rate is represented by a map $M(x,y)$, which is a function at each point on the target surface. The map M indicates a "resample factor", e.g., $M(x,y) = 1$ indicates a pixel that should progress at the same rate as the original captured sequence, $M(x,y) = 2$ at twice the original rate, $M(x,y) = 0.5$ at half the original rate, and so on. In the examples shown here, the length of the sequence is determined by the fastest pixel.

For each pixel in each frame of the output sequence, the source frame from the original sequence, $f_t(x,y)$, for that pixel is computed to implement the resampling:

$$f_t(x,y) = t \times M(x,y) \tag{8}$$

where t is the frame number in the output sequence, and $M(x,y)$ is the resample factor for that particular pixel.

The map M can be a function of any surface or environmental parameter. Ideally, the function would be data-driven, utilizing multiple datasets that track the rate of change versus temperature or surface curvature, for example. For the purpose of demonstration, however, we compute M using simple phenomenological expressions, that allows for easy control by the user. Figure 9 shows three examples where the synthesized TTF is controlled according to an outside parameter. First (a), the growth rate of grass is controlled by the intensity of light falling on the surface:

$$M = (\frac{1}{r^2}\cos\theta)^2 \tag{9}$$

where r is the distance to the source and θ is the angle of the source to the surface normal at pixel (x,y). The function is squared to provide a more dramatic falloff.

Next (b), the weathering rate of paint is controlled by the curvature of the surface. In this example, the curvature changes in only one dimension, so we can consider the surface as a collection of plane curves, each having the form of a sigmoid function:

$$y(x) = \frac{1}{1+e^{-x}} \tag{10}$$

The curvature is then computed using:

$$M = \frac{|\frac{d^2y}{dx^2}|}{|1+(\frac{dy}{dx})^2|^{\frac{3}{2}}} \tag{11}$$

Finally (c), the accumulation of snow on slate is controlled by the user defined map shown, that is simply the siggraph logo. M is a grayscale texture map with values ranging from 0 to 1, which determines the rate at which snow accumulates on the surface. Thus, the falling snow spells out the siggraph logo over time.

In Figure 10, a soccer ball is composited on grass and its shadow map is used to control the grass growth TTF. Grass growth occurs at a slower rate in the shadowed region. An image of the soccer ball was captured at a viewing angle

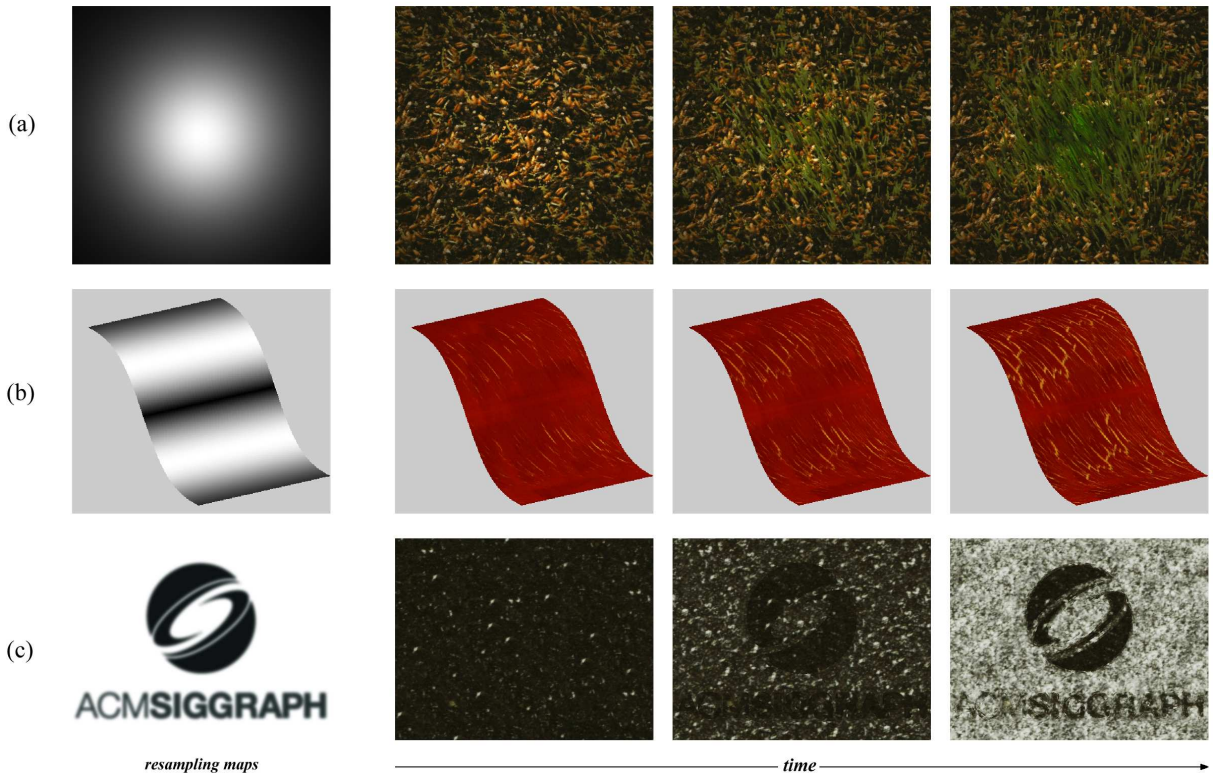

Figure 9: *Time-varying textures can be controlled simply using images that correspond to resampling maps (leftmost column) specifying the rate of evolution for different pixels. The maps are determined phenomenologically based on physical considerations or user input such as (a) light source intensity, (b) curvature, and (c) manually specified. The brighter regions on the map evolve faster in the time-varying textures.*

Figure 10: *Our approach provides a number of ways to control time-varying textures to mimic the behavior of the physical world. In this case, the grass growth TTF is controlled using a shadow map, corresponding to the shadow of the soccer ball.*

close to that of the grass acquisition on a solid color background. The soccer ball and the shadow cast by it onto the background were extracted from the image and overlayed on the synthesized grass. The shadow map acts as the map M for resampling the TTF.

7. Conclusions and Future Work

We have introduced a new class of textures, namely time-varying texture functions (TTFs), that capture the appearance of surfaces as they evolve over time. This leads to a new capability for computer graphics rendering, to include the dynamic evolution of surfaces and scenes, going beyond traditional notions of static texture. Our contributions include a newly acquired dataset of time-lapse images for many natural processes, a texture synthesis algorithm suited to the specific needs of time-varying textures, and new methods for controlling the time variation and rate of progression of these textures based on environmental conditions and user input.

Future work includes further study of the idea of controllability. This would include acquiring physically based controllability parameters, i.e., textures as they change dependent on shape or environmental factors. For example, TTFs could be captured on more complicated geometry, and a 3D scan completed simultaneously to map the rate of change to surface normal or curvature. Another example would be capturing grass growth TTFs, such as that shown in this paper, but with varying soil content, illumination conditions, water levels, or temperatures. In acquiring the datasets used in this work, we found that small changes in the environment or initial conditions would lead to dramatic differences in the progression and final state of the TTF. Understanding the source of these changes would certainly be enlightening.

Finally, as we noted in table 1, the TTF is only one time-varying appearance representation. We are also interested in exploring time-varying BRDFs (TBRDFs) and the full time-varying bi-directional texture function or TBTF, that includes lighting and view variation as well. We predict that the study of time-varying appearance properties will be a subject of significant future interest.

References

[Ada03] ADAM J. A.: *Mathematics in Nature: Modeling Patterns in the Natural World*. Princeton University Press, Princeton, NJ, 2003.

[Bal99] BALL P.: *The Self-Made Tapestry: Pattern Formation in Nature*. Oxford University Press, New York, NY, 1999.

[CH93] CROSS M., HOHENBERG P.: Pattern formation out of equilibrium. *Reviews of Modern Physics 65* (1993), 851–1112.

[DeB97] DEBONET J.: Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH* (1997), pp. 361–368.

[DEWJ*99] DORSEY J., EDELMAN A., WANN JENSEN H., LEGAKIS J., PEDERSON H.: Modeling and rendering of weathered stone. In *SIGGRAPH* (1999).

[DVGNK99] DANA K., VAN GINNEKEN B., NAYAR S., KOENDERINK J.: Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics 18*, 1 (January 1999), 1–34.

[EF01] EFROS A., FREEMAN W.: Image quilting for texture synthesis and transfer. In *SIGGRAPH* (Los Angeles, CA, August 2001), pp. 341–346.

[EL99] EFROS A., LEUNG T.: Texture synthesis by non-parametric sampling. In *ICCV* (September 1999).

[HB95] HEEGER D., BERGEN J.: Pyramid-based texture analysis/synthesis. In *SIGGRAPH* (July 1995), pp. 229–238.

[KSE*03] KWATRA V., SCHODL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003* (July 2003).

[LLX*01] LIANG L., LIU C., XU Y., GUO B., SHUM H.: *Real-Time Texture Synthesis By Patch-Based Sampling*. Tech. Rep. MSR-TR-2001-40, Microsoft Research, March 2001.

[LM01] LEUNG T., MALIK J.: Representing and recognizing the visual appearance of materials using 3d textons. *Int. J. Computer Vision 43*, 1 (June 2001), 29–44.

[MDG01] MERILLOU S., DISCHLER J.-M., GHAZANFARPOUR D.: Corrosion: simulating and rendering. In *Proceedings of Graphics Interface* (Ottawa, Canada, 2001).

[Mei92] MEINHARDT H.: Pattern formation in biology: a comparison of models and experiments. *Reports on Progress in Physics 55* (1992), 797–849.

[ML93] MOSTAFAVI M., LEATHERBARROW D.: *On Weathering: The Life of Buildings in Time*. MIT Press, Cambridge, MA, 1993.

[PL90] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[PS00] PORTILLA J., SIMONCELLI E.: A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Computer Vision 40*, 1 (October 2000), 49–70.

[SDW01] SOATTO S., DORETTO G., WU Y.: Dynamic textures. In *Int. Conf. on Computer Vision* (Vancouver, Canada, July 2001), pp. 439–446.

[SP96] SZUMMER M., PICARD R.: Temporal texture modeling. In *Int. Conf. on Image Processing* (Lausanne, Switzerland, September 1996).

[SSSE00] SCHODL A., SZELISKI R., SALESIN D. H., ESSA I.: Video textures. In *SIGGRAPH* (New Orleans, Louisiana, July 2000), pp. 489–498.

[Sti02] STICHLER C.: *Grass Growth and Development*. Tech. Rep. SCS-2002-22a, Soil and Crop Sciences, Texas A&M University, 2002.

[Tur91] TURK G.: Generating textures on arbitrary surfaces using reaction-diffusion. In *SIGGRAPH* (July 1991), pp. 289–298.

[Tur01] TURK G.: Texture synthesis on surfaces. In *SIGGRAPH* (August 2001), pp. 347–354.

[TZL*02] TONG X., ZHANG J., LIU L., WANG X., GUO B., SHUM H.: Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH* (2002).

[WK91] WITKIN A., KASS M.: Reaction-diffusion textures. In *SIGGRAPH* (July 1991), pp. 299–308.

[WL00] WEI L., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH* (July 2000), pp. 355–360.

[WL01] WEI L., LEVOY M.: Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH* (August 2001), pp. 355–360.

[WZ02] WANG Y., ZHU S.: A generative method for textured motion: Analysis and synthesis. In *Proc. European Conf. on Computer Vision* (Copenhagen, June 2002).

[ZG04] ZELINKA S., GARLAND M.: Jump map-based interactive texture synthesis. *ACM Transactions on Graphics 23*, 4 (2004), 930–962.