# Animating Pictures with Stochastic Motion Textures

Yung-Yu Chuang[1] Dan B Goldman[1] Brian Curless[1] David H. Salesin[1,2] Richard Szeliski[2]

[1]University of Washington [2]Microsoft Research

(a) Japanese temple (b) Harbor (c) The Boat Studio (d) Argenteuil (e) Sunflower

**Figure 1** Sample input images we animate using our technique. The first two pictures are photographs of a Japanese Temple (a) and a harbor (b). The paintings shown in (c) and (d) are Claude Monet's *Le bateau atelier (The Boat Studio)* and *The Bridge at Argenteuil*. We also try our method on Van Gogh's *Sunflower* (e) to animate the flowers.

## Abstract

In this paper, we explore the problem of taking a still picture and making it move in convincing ways. In this paper, we limit our domain to scenes containing passive elements that respond to natural forces in some oscillatory fashion. We use a semi-automatic approach, in which a human user segments the scene into a series of layers to be individually animated. The automatic part of the approach works by synthesizing a "stochastic motion texture" using a spectral method — i.e., a filtered noise spectrum whose inverse Fourier transform is the motion texture. The motion texture is a time-varying 2D displacement map, which is applied to each layer. The resulting warped layers are recomposited, along with "in-painting" to fill any holes, to form the animated frames. The result is a video texture created from a single still image, which has the advantages of being more controllable and of generally higher image quality and resolution than a video texture created from a video source. We demonstrate the technique on a variety of photographs and paintings.

## 1 Introduction

When we view a photograph or painting, we perceive much more than the static picture before us. We supplement that image with our life experiences: given a picture of a tree, we imagine it swaying; given a picture of a pond, we imagine it rippling. In effect, we bring to bear a strong set of "priors" (to use the technical jargon from computer vision), and these priors enrich our perceptions.

In this paper, we explore how a set of explicitly encoded priors might be used to animate pictures on a computer. The *fully automatic* animation of *arbitrary* scenes is, of course, a monumental challenge. In order to make progress, we make the problem easier in two ways. First, we use a semi-automatic, user-assisted, approach. In particular, we have a user segment the scene into a set of animatable layers and assign certain parameters to each one. Second, we limit our scope to scenes containing passive elements that respond to natural forces in some oscillatory fashion. The types of passive elements we explore include plants and trees, water, floating objects like boats, and clouds. The motion of these objects is driven by the same natural force, namely, wind. While these may seem like a limited set of objects and motions, they occur in a large variety of pictures and paintings, as shown in Figure 1.

It turns out that all of these elements can be animated in a similar way. First, we segment the picture into a set of user-specified layers using Bayesian matting [Chuang et al. 2001]. As each layer is removed from the picture, "in-painting" is used to fill in the resulting hole. Next, for each layer, we synthesize a *stochastic motion texture* using spectral methods [Stam 1995]. Spectral methods work by generating a noise spectrum in the frequency domain; applying a (physically based) spectrum filter to that noise, which is specific to the type of natural force and to the type and parameters of the passive object being affected; and computing an inverse Fourier transform to create the stochastic motion texture. This motion texture is a time-varying 2D displacement map, which is applied to the pixels in the layer. Finally, the warped layers are recomposited to form the animated picture for each frame.

The resulting moving picture can be thought of as a kind of video texture [Schödl et al. 2000]—although, in this case, a video texture created from a single static image rather than from a video source. Thus, these results have potential application wherever video textures do, i.e., in place of still images on Web sites, as screen savers or desktop "wallpapers", or in presentations and vacation slide shows. In addition, there are several advantages to creating video textures from a static image rather than from a video source.

First, because they are created synthetically, they allow greater creative control in their appearance. For example, the wind direction and amplitude can be tuned for a particular desired effect. Second, consumer-grade digital still cameras generally provide much higher image quality and greater resolution than their videocamera counterparts. This set of advantages may allow video textures to be used in entirely new situations that were not previously practical. For example, controllable, high-resolution video textures might be usable for animated matte paintings in special effects.

For the most part, the algorithms we describe in this paper are fairly simple applications of already available techniques. Thus, perhaps the paper's greatest contributions are its formulation of the overall problem, its introduction of the concept of stochastic motion textures, and its proof of the feasibility of applying a warping-based approach to creating surprisingly convincing and appealing animated pictures.

### 1.1 Related work

Our goal is to synthesize a stochastic video from a single image. Hence, our work is directly related to the work on video textures and dynamic textures [Szummer and Picard 1996; Schödl et al. 2000; Wei and Levoy 2000; Soatto et al. 2001; Wang and Zhu 2003]. Like

our work, video textures focus on "quasi-periodic" scenes. However, the inputs to video texture algorithms are short videos that can be analyzed to mimic the appearance and dynamics of the scene. In contrast, the input to our work is only a single image.

Our work is, in spirit, similar to the "Tour Into the Picture" system developed by Horry *et al.* [1997]. Their system allows users to map a 2D image onto a simple 3D box scene based on some interactively selected perspective viewing parameters such as vanishing points. This allows users to interactively navigate into a picture. Criminisi *et al.* [2000] propose an automated technique that can produce similar effects in a geometrically correct way. More recently, Oh *et al.* [2001] developed an image-based depth editing system capable of augmenting a photograph with a more complicated depth field to synthesize more realistic effects. In our work, instead of synthesizing a depth field to change the viewpoint, we add motion fields to make the scene change over time.

For certain classes of motions, our system requires the users to specify a skeleton for a layer. It then performs a physically-based simulation on the skeleton to synthesize a motion field; it is therefore similar to *skeleton-based animation* approaches. Litwinowicz and Williams [1994] use keyframe line drawings to deform images to create 2D animations. Their system is quite useful for traditional 2D animation. However, their technique is not suitable for modeling the natural phenomena we target because such motions are difficult to keyframe. Also, they use a smooth scattered data interpolation to synthesize motion field without any physical dynamics model.

Our work also has similar components to the *object-based image editing* system proposed by Barrett and Cheney [2002], namely, *object selection*, *matte extraction*, and *hole filling*. Indeed, Barrett *et al.* have also demonstrated how to generate a video from a single image by editing and interpolating keyframes. Like Litwinowicz's system, the focus is on key-framed rather than stochastic (temporal texture-like) motions.

An earlier attempt to create the illusion of motion from an image was the "Motion without movement" paper by Freeman *et al.* [1991]. They apply quadrature pairs of oriented filters to vary the local phase in an image to give the illusion of motion. While the motion is quite compelling, the band-pass filtered images do not look photorealistic.

Even earlier, at the turn of the (20th) century, people painted outdoor scenes on pieces of masked vellum paper and used series of sequentially timed lights to create the illusion of descending waterfalls [Hathaway et al. 2003]. People still make this kind of device, which is often called a *kinetic waterfall*. Another example of a simple animated picture is the popular Java program *Lake applet*, which takes a single image and perturbs the image with a set of simple ripples [Griffiths 1997]. Though visually pleasing, these results often do not look realistic because of their lack of physical properties.

Working on an inverse problem to ours, Sun *et al.* [2003] propose a *video-input driven animation* (VIDA) system to extract physical parameters, like wind speed, from real video footage. They then use these parameters to drive the physical simulations of synthetic objects to integrate them consistently with the source imagery. They estimate physical parameters from observed displacements; we synthesize displacements using a physical simulation based on user-specified parameters. They target a similar set of natural phenomena to those we study, such as plants, waves, and boats, which can all be explained as *harmonic oscillations*.

To simulate our dynamics, we use physically-based simulation techniques previously developed in computer graphics for modeling natural phenemena. For waves, we use the Fourier wave model to synthesize a time-varying height field. Mastin *et al.* [1987] were the first to introduce statistical frequency-domain wave models from oceanography into computer graphics. In a similar way, we synthesize stochastic wind fields [Shinya and Fournier 1992; Stam and Fiume 1993] by applying a different spectrum filter. When applying the wind field to trees, since the force is oscillatory in nature, the responding motions are also periodic and can be solved more robustly and efficiently in the frequency domain [Stam 1997; Shinya et al. 1998].

Aoki *et al.* [1999] coupled physically-based animations of plants with image morphing techniques as an efficient alternative to the expensive physically-based plant simulation and synthesis and therefore only demonstrate their concept on synthetic images. In our work, we target real pictures and use our approach as a way to synthesize video textures for stochastic scenes.

Our system requires users to segment an image into *layers*. To support seamless composites, a soft alpha matte for each layer is required. We use recently proposed interactive image matting algorithms to extract alpha mattes from the input image [Ruzon and Tomasi 2000; Chuang et al. 2001]. To fill in holes left behind after removing each layer, we use an *inpainting* algorithm [Bertalmio et al. 2000; Criminisi et al. 2003; Jia and Tang 2003; Drori et al. 2003].

### 1.2 Overview

We begin with a system overview that describes the basic flow of our system (Section 2). We then address our most important subproblem, namely synthesizing stochastic motion texture (Section 3). Finally, we discuss our results (Section 4) and end with conclusions and ideas for future work.

## 2 System overview

Given a single image $I$, how can we generate a continuously moving animation? The approach we follow is to break the image up into several layers and to then synthesize a *motion texture*[1] and apply it to each layer individually.

A motion texture is essentially a *time-varying displacement map* defined by a motion type, a set of motion parameters, and optionally a motion skeleton. A displacement map $D$ is a set of displacement vectors,

$$D(p) = (d_x(p), d_y(p)) \tag{1}$$

for pixels $p = (x, y)$. A motion texture $M(t)$ is a mapping from a time $t$ to a displacement map $D$.

Applying a displacement field $D$ directly to an image $I$ results in a forward warped image $I'$ such that

$$I'(x + d_x(p), y + d_y(p)) = I(x, y). \tag{2}$$

However, since forward mapping is fraught with problems such as aliasing and holes, we actually use inverse warping, $I' = D' \star I$, where

$$I'(x, y) = I(x + d'_x(p), y + d'_y(p)). \tag{3}$$

We could compute the inverse displacement map $D'$ from $D$ using the two-pass method suggested in [Shade et al. 1998]. Instead, since our motion fields are all very smooth, we simply dilate them by the extent of the largest possible motion and reverse their sign.

With this notation in place, we can now describe the basic workflow of our system (Figure 2), which consists of three steps: *layering* and *matting*, *motion specification* and *editing*, and finally *rendering*.

[1] We use the terms *motion texture* and *stochastic motion texture* interchangeably in the paper. The term *motion texture* was also used in [Li et al. 2002] to refer a linear dynamic system learned from motion capture data.

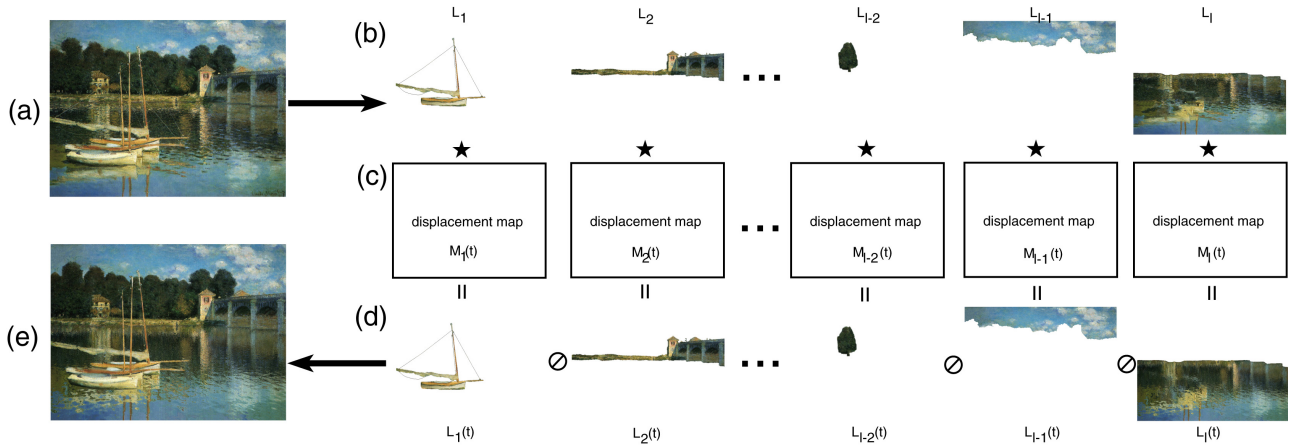


**Figure 2** Overview of our system. The input still image (a) is manually segmented into several layers (b). Each layer $L_i$ is then animated with a different stochastic motion texture $M(t)$ (c). Finally, the animated layers $L_i(t)$ (d) are composited back together to produce the final animation $I(t)$ (e).

**Layering and matting.** The first step, *layering*, is to segment the input image $I$ into layers so that, within each layer, the same motion texture can be applied. For example, for the painting in Figure 2(a), we have the following layers: water, sky, bridge and shore, three boats, and the eleven trees in the background (Figure 2(b)). To accomplish this, we use an interactive object selection tool such as a painting tool or intelligent scissors [Mortensen and Barrett 1995]. The tool is used to specify a *trimap* for a layer; we then apply Bayesian matting to extract the color image and a soft alpha matte for that layer [Chuang et al. 2001].

Because some layers will be moving, occluded parts of the background might become visible. Hence, after extracting a layer, we use an inpainting algorithm to fill the hole in the background behind the foreground layer. We use the example-based inpainting algorithm of [Criminisi et al. 2003] because of its simplicity and its capacity to handle both linear structures and textured regions. Note that the inpainting algorithm does not have to be perfect, since only pixels near the boundary of the hole are likely to become visible. However, sometimes, we do have to perform manual inpainting to better maintain layers' structures, for example, the boats in Figure 2. After the background image has been inpainted, we work on this image to extract the next layer. We repeat this process from the closest layer to the furthest layer to generate the desired number of layers. Each layer $L_i$ contains a color image $C_i$, a matte $\alpha_i$, and a depth $d_i$. The depth could be automatically assigned with the order in which the layers are extracted.

**Motion specification and editing.** The second component of our system lets us *specify* and *edit* the motion texture for each layer. Currently, we provide the following motion types: *trees* (swaying), *water* (rippling), *boat* (bobbing), *clouds* (translation), and *stone* (no motion :-). For each motion type, the user can tune the motion parameters and specify a motion skeleton, where applicable. We describe the motion parameters and skeletons in more details for each motion type in Section 3.

Since all of the motions we currently support are driven by wind, the user controls a single wind speed and direction, which is shared by all the layers. This allows all the layers to respond to the wind consistently. Our motion synthesis algorithm is fast enough to animate a single layer in real-time. Hence, the system can provide instant visual feedback to changes in motion parameters, which makes motion editing easier. Each layer $L_i$ has its own motion texture, $M_i$, as shown in Figure 2(c).

**Rendering.** During the *rendering* process, for each time instance $t$ and layer $L_i$, a displacement map $M_i(t)$ is synthesized. This displacement map is then applied to $C_i$ and $\alpha_i$ to obtain $L_i(t) = M_i(t) \star L_i(0)$ (Figure 2(d)). Notice that the displacement is evaluated as an absolute displacement of the input image $I(0)$ rather than a relative displacement of the previous image $I(t-1)$. In this way, repeated resampling is avoided.

Finally, all the warped layers are composited together from back to front to synthesize the frame at time $t$, $I(t) = L_1(t) \oslash L_2(t) \oslash \ldots \oslash L_l(t)$, where $d_1 \geq d_2 \cdots \geq d_l$ and $\oslash$ is the standard *over* operator [Porter and Duff 1984] (Figure 2(e)). The user can also specify a time-varying wind field to create a more realistic animation.

## 3 Stochastic motion textures

In this section, we describe our approach to synthesizing the stochastic motion textures that drive the animated image. In Section 3.1, we describe the basic principles (spectral methods). We then describe the details of each motion type, i.e., trees (Section 3.2), water (Section 3.3), bobbing boats (Section 3.4), and clouds (Section 3.5).

### 3.1 Stochastic modeling of natural phenomena

Many natural motions can be seen as harmonic oscillations [Sun et al. 2003], and, indeed, hand-crafted superpositions of handfuls of sinusoids have often been used to approximate many natural phenomena for computer graphics. However, this simple approach has some limitations. First of all, it is tedious to tune the parameters to produce the desired effects. Second, it is harder to hook all the motions in a consistent way since they lack a physical basis. Lastly, the resulting motions do not look natural since they are strictly periodic — irregularity actually plays a central role in modeling natural phenomena.

One way to add randomness is to introduce a noise field. Introducing this noise directly into the temporal or spatial domain often leads to erratic and unrealistic simulations of natural phenomena. Instead, we simulate noise in the frequency domain, and then sculpt the spectral characteristics to match the behaviors of real systems that have intrinsic periodicities and frequency responses. Specific spectrum filters need to be applied to model specific phenomena, leading to so-called *spectral methods*.

The spectral method for synthesizing a stochastic field in general has three steps: (1) generate a complex Gaussian random field in the frequency domain, (2) apply a domain-specific spectrum filter, (3) compute the inverse Fourier transform to synthesize a stochastic field in the time or frequency domain. A nice property of this method is that the synthesized stochastic field can be tiled seamlessly. Hence, we only need to synthesize a patch of reasonable size

and tile it to produce a much larger stochastic signal. This tiling approach works reasonably well if the size of the patch is large enough to avoid objectionable repetition.

To realistically model natural phenomena, the filter should be learned from the real-world data. For the phenomena we simulate, plants and waves, such experimental data and statistics are available from other fields, e.g., structural engineering and oceanography, and have already been used by the graphics community to create synthetic imagery [Shinya and Fournier 1992; Stam and Fiume 1993; Mastin et al. 1987]. We use these methods to synthesize our stochastic motion textures in the following sections.

### 3.2 Plants and trees

The branches and trunks of trees and plants can be modeled as physical systems with mass, damping, and stiffness properties. The driving function that causes branches to sway is typically wind. Our goal is to model the spectral filtering due to the dynamics of the branches applied to the spectrum of the driving wind force.

To model the physics of branches, we take a simplified view introduced by [Sun et al. 2003]. In particular, each branch is represented as a 2D line segment parameterized by $u$, which ranges from 0 to 1. This line segment is drawn by the user for each layer. Displacements of the tip of the branch $d_{\text{tip}}(t)$ are taken to be perpendicular to the line segment. Modal analysis indicates that the displacement perpendicular to the line for other points along the branch can be simplified to the form:

$$d(u,t) = \left[\frac{1}{3}u^4 - \frac{4}{3}u^3 + 2u^2\right] d_{\text{tip}}(t) \tag{4}$$

We approximate the (scalar) displacement of the tip in the direction of the projected wind force as a dampled harmonic oscillator:

$$\ddot{d_{\text{tip}}}(t) + \gamma \dot{d_{\text{tip}}}(t) + 4\pi^2 f_o^2 d_{\text{tip}}(t) = r(t)/m \tag{5}$$

where $m$ is the mass of the branch, $f_o = k/m$ is the natural frequency of the system, $\gamma = c/m$ is the velocity damping term [Sun et al. 2003]. These parameters have a more intuitive meaning than the damping ($c$) and stiffness ($k$) terms found in more traditional formulations. The driving force $r(t)$ is derived from the wind force incident on the branch, as detailed below.

Taking the temporal Fourier transform of this equation gives us:

$$D_{\text{tip}}(f) = \frac{R(f)\exp^{-i2\pi\theta}}{2\pi m\left[\left(f^2 - f_o^2\right)^2 + \gamma^2 f^2\right]^{-1/2}} \tag{6}$$

where $R(f)$ is the Fourier transform of the driving force. The phase shift $\theta$ is given by:

$$\tan\theta = \frac{\gamma f}{f^2 - f_o^2} \tag{7}$$

We can see from equation 6 that the dynamical system is acting as a non-zero phase spectral filter on the forcing spectrum $R(f)$.

Next, we model the forcing spectrum for wind. Experimental evidence [Simiu and Scanlan 1986, p. 55] indicates that the temporal velocity spectrum of wind at a point takes the following form:

$$V(f) \sim \frac{v_{\text{mean}}}{\left(1 + \kappa f / v_{\text{mean}}\right)^{5/3}} \tag{8}$$

where $v_{\text{mean}}$ is the mean wind speed and $\kappa$ is generally a function of altitude which we take to be a constant. We therefore modulate a random Gaussian noise field $G(f)$ with the velocity spectrum to compute the spectrum of a particular (random) wind velocity field:

$$\tilde{V}(f) = V(f)G(f) \tag{9}$$

The force due to the wind is generally modeled as a drag force proportional to $\tilde{v}^2(t)$. However, in our experiments, we have found that making the wind force directly proportional to wind velocity produces more pleasing results.

Finally, we assemble equations 6-9 to construct the spectrum of the tip displacement $D_{\text{tip}}(f)$, take the inverse Fourier transform to generate the tip displacement, $d_{\text{tip}}(t)$, and distribute the displacement over the branch according to equation 4. The displacement of points in the layer away from the skeleton is obtained by projecting all pixels orthogonally onto the original skeleton and using the corresponding displacement.

The user can control the resulting motion appearance by independently changing the mean wind speed $v_{\text{mean}}$ and the natural (oscillatory) frequency $f_o$, mass $m$, and velocity damping term $\gamma$ of each branch.

### 3.3 Water

Water surfaces belong to another class of natural phenomena that exhibit oscillatory responses to natural forces like wind. In this section we describe how one can specify a water plane in a photograph and then define the mapping of water height out of that plane to displacements in image space. We then describe how to synthesize water height variations, again using a spectral method.

The motion skeleton for water is simply a plane; we assume that the image plane is the *x-y plane* and the water surface is parallel to the *x-z plane*. To correctly model the perspective effect, the users roughly specifies where the plane is. This perspective transformation $T$ can be fully specified by the focal length and the tilt of the camera, which can be visualized by drawing the horizon [Criminisi et al. 2000].

After specifying the water plane, the water is animated using a time-varying height field $h(q,t)$, where $q = (x_q, y_0, z_q)$ is a point on the water plane. To convert the height field $h$ to the displacement map $M_t(p)$, for each pixel $p$ we first find its corresponding point, $q = (x_q, y_0, z_q) = Tp$, on the water plane. We then add the synthesized height $h(q,t)$ as a vertical displacement, which gives us a point, $q' = (x_q, y_0 + h(q,t), z_q)$. We then project $q'$ back to the image plane to get $p' = T^{-1}q'$. The displacement vector for $M_t(p) = p' - p$ is therefore:

$$M_t(p) = T^{-1}[Tp + (0, h(Tp,t), 0)] - p \tag{10}$$

The above model is technically correct if we want to displace objects on the surface of the water. In reality, the shimmer in the water is caused by local changes in surface normals. Therefore, a more physically realistic approach would be to use *normal* mapping, i.e., to convert the surface normals computed from the spatial gradients of $h(q,t)$ into two-dimensional displacements. We have found that our current approach produces pleasing, realistic-looking results and plan to study the more physically-motivated reflections model in the future.

To synthesize a time-varying height field for the water, we use the time-varying wind velocity derived in the previous section to synthesize a height field matching the statistics of real waves, as described by Mastin *et al.* [1987].

The spectrum filter we use for waves is the *Phillips spectrum* [Tessendorf 2001], which is a power spectrum describing the expected square amplitude of waves across all spatial frequencies, $\mathbf{s}$:

$$P(\mathbf{s}) \sim \frac{\exp\left[-1/\left(sL\right)^2\right]}{s^4} \left|\hat{\mathbf{s}} \cdot \hat{\mathbf{v}}_{\text{mean}}\right|^2 \tag{11}$$

where $s = |\mathbf{s}|$, $L = v_{\text{mean}}^2/g$, $g$ is the gravitational constant and $\hat{\mathbf{s}}$ and $\hat{\mathbf{v}}_{\text{mean}}$ are normalized spatial frequency and wind direction vectors.

The square root of the power spectrum describes the amplitude of wave heights, which we can use to filter a random Gaussian noise field:

$$H_0(\mathbf{s}) = a\sqrt{P(\mathbf{s})}G(\mathbf{s}) \quad (12)$$

where $a$ is a constant of proportionality and $H_0$ is an instance of the height field which we can now animate by introducing time-varying phase. However, waves of different spatial frequencies move at different speeds. The relationship between the spatial frequency and the phase velocity is described by the well-known dispersion relation,

$$w(s) = \sqrt{gs}. \quad (13)$$

The time varying height spectrum can thus be expressed as:

$$H(\mathbf{s}, t) = H_0(\mathbf{s}) \exp[iw(s)t] + H_0^*(-\mathbf{s}) \exp\{-iw(s)t\} \quad (14)$$

where the $H_0^*$ is the complex conjugate of $H_0$. We can now compute the height field at time, $h(q, t)$ as the two-dimensional inverse Fourier transform of $H(\mathbf{s}, t)$ with respect to spatial frequencies $\mathbf{s}$. We take the generated height field and tile the water surface using a scale parameter, $\beta$, to control the spatial frequency.

There are thus several motion parameters related to water: wind speed, wind direction, the size of the tile $N$, the amplitude scale $a$, and the spatial frequency scale $\beta$. The wind speed and direction are controlled globally for the whole animation. We find that a tile of size $N = 256$ usually produces nice looking results. Users can change $a$ to scale the height of the waves/ripples. Finally, scaling the frequencies by $\beta$ changes the scale at which the wave simulation is being done. Simulating at a larger frequency scale gives a rougher look, while a smaller scale gives a smoother look. Hence, we call $\beta$ the *roughness* in our user interface.

### 3.4 Boats

We approximate the motion of a bobbing boat by an 2D rigid transformation composed of a translation for heaving and a rotation for rolling. A boat moving on the surface of open water is almost always in oscillatory motion [Sun et al. 2003]. Hence, the simplest model is to assign a sinusoidal translation and a sinusoidal rotation. However, this often looks fake. In principle, we could build a simple model for the boat, convert the height field of water into a force interacting with the hull and solve the dynamics equation for the boat to estimate its displacement. However, since our goal is only to synthesize an approximate solution, we directly use the height field of the wave to move the boat, as follows.

We let the user select a line close to the bottom of the boat. Then, we sample several points along the line. For each point $q_i$, we look up the corresponding 2D projection of the height field $h(q_i, t)$. Finally, we use linear regression to fit a line through the $q_i$'s. The position and orientation of the fitted line then determine the heaving and rolling of the boat.

### 3.5 Clouds

Another common element for scenic pictures is clouds. In principle, clouds could also be modeled as a stochastic process. However, we need the stochastic process to match the clouds in the image at some point, which is harder. Since clouds often move very slowly and their motion does not attract too much attention, we simply assign a translational motion field to them. As with water, we have to extend the clouds outside the image frame in some way, since their motion in one direction will create holes that we have to fill.

## 4 Results

We have developed an interactive system that supports matting, inpainting, motion editing, and previewing the results. We have applied our system to several photographs and famous paintings. The final results are best viewed in video form (http://grail.cs.washington.edu/projects/StochasticMotionTextures). Here, we summarize some of the results and discuss some details in creating them.

It takes from several minutes to several hours to animate a picture depending on the complexity of the input. Matting and inpainting is the most time-consuming part in our system and usually consumes more than 90 percent of the overall time for animating a picture. For example, for the picture in Figure 1(a), because of the complicated structure, it is difficult to specify the trimap for the branches on the left. Since the background is smooth in color, we end up using a garbage matte to take out the tree and use inpainting algorithm first to estimate the background. Taking advantage of the known background, the trimap can be painted fairly roughly. We model a total of 10 branches on the left and the right. We use a small wave amplitude and high roughness to give the ripples a fine-grained look. For the harbor picture in Figure 1(b), we animate the water and have nine boats swing with the water. The cloud and sky are animated using a translational motion field.

Figure 1(c)-(e) shows three paintings we have animated. Our technique actually works even better with paintings, perhaps because in this situation we are less sensitive to anything that does not look perfectly realistic. For Claude Monet's painting in Figure 1(c), we animate the water with lower amplitude roughness to keep the strokes intact. We also let the boat sway with the water. Another Monet's painting shown in Figure 1(d) is a more complex example, with more than twenty layers. We use this example to demonstrate that we can change the appearance of the water by controlling the physical parameters. In Figure 3, we show look of the water under different wind speeds, directions, and simulation scales.

For Van Gogh's sunflower painting (Figure 1(e)), we use our stochastic wind model to animate the forty plant layers. With a simple sinusoidal model, the viewer usually can quickly figure out that the plants swing in synchrony and the motion looses a lot of its interest. With the stochastic wind model, the flowers' motions decorrelate in phase and the resulted animation is more appealing.

## 5 Conclusion and future work

In this paper, we have described an approach for animating still pictures of a limited class of scenes—in particular, scenes that contain passive elements responding to natural forces in an oscillatory fashion. We envision this work as just the first step in the larger problem of animating a much more general class of pictures.

Even for the scenes we can already animate, our system currently makes a number of assumptions that we would like to relax. For example, we assume that the elements of the input image are in their equilibrium positions. This is often not the case in reality — for instance, for a scene with water that already has ripples. Indeed, an interesting challenge would be to use these ripples to estimate the water motion and then animate it correctly. (In addition, rippling water also alternates between acting as a transmitter and as a reflector near the Fresnel angle. This effect should also be incorporated.) As another example, our method currently works best for trees at a distance. For closer trees, it is difficult and tedious to segment the trees properly. It would also be interesting to add shimmering inside the trees to simulate leaf motion using some kind of turbulent flow fields.

There are other classes of motion that could be modeled using a similar approach. For example, waterfalls could perhaps be animated using a technique similar to "motion without movement" [Freeman et al. 1991]. Ocean waves could be simulated using stochastic models, although matching the appearance of the still poses some interesting challenges. Flying birds and other small animals could be animated using ideas from video sprites [Schödl et al.

(a) composite (b) lower wind speed (c) wind of different direction (d) rougher water surface

**Figure 3** We can control the appearance of water surface by adjusting some physical parameters such as wind speed. We show one of the composites (a) as the reference, in which the wind blow at 5 m/s in $z$ direction. We decrease the wind speed to 3 m/s (b) and change the wind direction to be along $z$ axis (c). In (d), we change the scale of the simulation to render water with finer ripples.

2000]. We believe that it might also be possible to animate fluids like flame or smoke. However, this requires a constrained stochastic simulation, since the state of simulation should resemble the appearance of the input image. Recent advances in controlling smoke simulation by keyframes could be used for this purpose [Treuille et al. 2003].

In our system, all the layers are hooked up together to a synthetic wind force. Currently, the same wind force is applied everywhere in the scene, albeit with different phases. It should not be too difficult to extend the formulation to handle a complete vector fields of evolving wind forces in order to provide a more realistic style of animation. In addition, we would like to add more controllability so that the users could possibly interact with the trees individually.

Currently, we use physically-based simulation to synthesize a parametric motion field. One way to improve the quality of the motion would be to use learning algorithms to transfer motion from similar type of objects in videos. Indeed, the physically-based approaches we use are based on empirically derived models. Learning the motion directly at the pixel level could give finer-gained motions. This approach also might provide a type of "canned" motion library that could be used to paint motion fields interactively onto a scene.

Our system now requires a fair amount of user interaction. For example, to model a tree realistically, the user has to manually segment a tree into clumps and then draw a skeleton to connect them. One way to reduce this burden would be to let the user sketch branches on the tree, and let the system automatically figure out how the tree should be subdivided and associated with the skeleton.

Another possibility would be to use multiple pictures as input. Most modern digital cameras have a "motor-drive" mode that allows users to take high-resolution photographs at a restricted sampling rate, around 1–3 frames per second. From such a set of photographs, we might be able to automatically segment a picture into several coherently moving regions and figure out the motion parameters from the sample still images. It may also be interesting to combine high-resolution stills with lower-resolution video to produce attractive animations. Our approach could also be combined with "Tour into the picture" to provide even richer control over animated pictures.

In conclusion, we are pleased by the ease with which it is possible to breathe life into pictures, based on recent improvements in matting, in-painting, and stochastic modeling algorithms and we feel like we have only just begun to explore the creative possibilities in this rich domain.

## References


AOKI, M., SHINYA, M., TSUTSUGUCHI, K., AND KOTANI, N. 1999. Dynamic texture: Physically-based 2d animation. In *Siggraph 1999 Conference Sketches and Applications*, 239.

BARRETT, W. A., AND CHENEY, A. S. 2002. Object-based image editing. In *Proceedings of ACM SIGGRAPH 2002*, 777–784.

BERTALMIO, M., SAPIRO, G., CASELLES, V., AND BALLESTER, C. 2000. Image inpainting. In *Proceedings of SIGGRAPH 2000)*, 417–424.

CHUANG, Y.-Y., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2001. A Bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001*, vol. 2, 264–271.

CRIMINISI, A., REID, I. D., AND ZISSERMAN, A. 2000. Single view metrology. *International Journal of Computer Vision 40*, 2, 123–148.

CRIMINISI, A., PEREZ, P., AND TOYAMA, K. 2003. Object removal by exemplar-based inpainting. In *Proceedings of IEEE CVPR 2003*, vol. II, 721–728.

DRORI, I., COHEN-OR, D., AND YESHURUN, H. 2003. Fragment-based image completion. *ACM Transactions on Graphics 22*, 3, 303–312.

FREEMAN, W. T., ADELSON, E. H., AND HEEGER, D. J. 1991. Motion without movement. In *Proceedings of ACM SIGGRAPH 1991*, vol. 25, 27–30.

GRIFFITHS, D., 1997. Lake java applet.

HATHAWAY, T., BOWERS, D., PEASE, D., AND WENDEL, S., 2003. http://www.mechanicalmusicpress.com/history/pianella/p40.htm.

HORRY, Y., ANJYO, K.-I., AND ARAI, K. 1997. Tour into the picture: using a spidery mesh interface to make animation from a single image. In *Proceedings of ACM SIGGRAPH 1997*, 225–232.

JIA, J., AND TANG, C.-K. 2003. Image repairing: Robust image synthesis by adaptive nd tensor voting. In *Proceedings of IEEE CVPR 2003*, vol. I, 643–650.

LI, Y., WANG, T., AND SHUM, H.-Y. 2002. Motion texture: a two-level statistical model for character motion synthesis. In *Proceedings of ACM SIGGRAPH 2002*, 465–472.

LITWINOWICZ, P., AND WILLIAMS, L. 1994. Animating images with drawings. In *Proceedings of ACM SIGGRAPH 1994*, 409–412.

MASTIN, G. A., WATTERBERG, P. A., AND MAREDA, J. F. 1987. Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Applications 7*, 3 (1987), 16–23.

MORTENSEN, E. N., AND BARRETT, W. A. 1995. Intelligent scissors for image composition. In *Proceedings of ACM SIGGRAPH 1995*, 191–198.

OH, B. M., CHEN, M., DORSEY, J., AND DURAND, F. 2001. Image-based modeling and photo editing. In *Proceedings of ACM SIGGRAPH 2001*, 433–442.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. In *Proceedings of ACM SIGGRAPH 1984*, 253–259.

RUZON, M. A., AND TOMASI, C. 2000. Alpha estimation in natural images. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 18–25.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *Proceedings of ACM SIGGRAPH 2000*, 489–498.

SHADE, J., GORTLER, S., HE, L.-W., AND SZELISKI, R. 1998. Layered depth images. In *Proceedings of ACM SIGGRAPH 1998*, 231–242.

SHINYA, M., AND FOURNIER, A. 1992. Stochastic motion – motion under the influence of wind. *Computer Graphics Forum 11*, 3, 119–128.

SHINYA, M., MORI, T., AND OSUMI, N. 1998. Periodic motion synthesis and fourier compression. *The Journal of Visualization and Computer Animation 9*, 3, 95–107.

SIMIU, E., AND SCANLAN, R. H. 1986. *Wind Effeccts on Structures*. John Wiley & Sons.

SOATTO, S., DORETTO, G., AND WU, Y. N. 2001. Dynamic textures. In *Proceedings of IEEE ICCV '01*, 439–446.

STAM, J., AND FIUME, E. 1993. Turbulent wind fields for gaseous phenomena. In *Proceedings of ACM SIGGRAPH 1993*, 369–376.

STAM, J. 1995. *Multi-Scale Stochastic Modelling of Complex Natural Phenomena*. PhD thesis, Dept. of Computer Science, University of Toronto.

STAM, J. 1997. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Computer Graphics Forum 16*, 3, 159–164.

SUN, M., JEPSON, A. D., AND FIUME, E. 2003. Video input driven animation (VIDA). In *Proceedings of IEEE ICCV 2003*, 96–103.

SZUMMER, M., AND PICARD, R. W. 1996. Temporal texture modeling. In *IEEE ICIP 1996*, vol. 3, 823–826.

TESSENDORF, J. 2001. Simulating ocean water. *Siggraph course notes*.

TREUILLE, A., MCNAMARA, A., POPOVIC, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Trans. Graph. 22*, 3, 716–723.

WANG, Y., AND ZHU, S. C. 2003. Modeling textured motion: Particle, wave and sketch. In *Proceedings of IEEE ICCV 2003*, 213–220.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. *Proceedings of ACM SIGGRAPH 2000*, 479–488.