Eurographics Symposium on Rendering (2004)
H. W. Jensen, A. Keller (Editors)

A spectral-particle hybrid method for rendering falling snow

M. S. Langer, L. Zhang, A.W. Klein, A. Bhatia, J. Pereira, D. Rekhi

School of Computer Science, McGill University, Montreal, Canada

Abstract

Falling snow has the visual property that it is simultaneously a set of discrete 3D particles and a 3D dynamic texture. Capturing the texture properties of falling snow using standard particle systems can, however, require enough particles to severely impact rendering rates. Here we show how to address this limitation by rendering the texture properties directly. We use a standard particle system to generate a relatively sparse set of falling snow flakes for a given scene, and we then composite in a dynamic texture. The texture is generated using a novel image-based spectral synthesis method. The spectrum of falling snow is defined by a dispersion relation in the image plane, derived from linear perspective. The dispersion relation relates image speed, image size, and snowflake depth. In the frequency domain, it relates the wavelength and speed of moving 2D image sinusoids. The parameters of this spectral snow can be varied both across the image and over time. This provides the flexibility to match the direction and speed parameters of the spectral snow to those of the falling particle snow. Camera motion can also be matched. Our method produces visually pleasing results at interactive rendering rates. We demonstrate our approach by adding snow effects to static and dynamic scenes. An extension for creating rain effects is also presented.

1. Introduction

Falling snow turns a bleak winter scene into a romantic wonderland. Its charm has inspired many artists, as well as many computer graphics researchers.

Most previous computer graphics methods for rendering snow have been based on 3D particle systems. These may be divided into two types: those that render static images of *fallen* snow [Fea00, SOH99, NIDN97], and those that render animated imagery of *falling* snow [Ree83, Sim90]. This paper is concerned with the latter problem.

In addition, several methods have been developed for modeling the global forces that drive particles such as snow along their paths [Sta01, SF92, SF93, EMP*03]. This paper is not concerned with simulating these physical forces, as the methods just cited do an admirable job already. Instead, this paper addresses a different and neglected aspect of the problem of rendering falling snow.

An intriguing property of falling snow is its particle-texture dual nature. Snowflakes are obviously particles. Individual snowflakes are clearly visible when we observe falling snow. At the same time, one perceives not just the snowflakes, but also the large scale forces that drive them as they fall and swirl in group motion. These larger scale percepts arise from Gestalt-like configural relationships between the snowflakes as they move. Falling snow generates a flow pattern, which has a wholeness beyond the individual particles. The dynamic texture created by these flow patterns is the result of hundreds of thousands of snowflakes being present in a real scene. If one could model falling snow using a particle system with hundreds of thousand of particles, then one would obtain the dynamic texture properties of falling snow automatically. Unfortunately, as we will see, the computational cost of doing so can be quite expensive.

Therefore, in this paper, we introduce a method that keeps the number of particles relatively low, and that achieves the texture properties of falling snow using a novel *image-based spectral synthesis* method. The spectral synthesis method produces a dynamic texture which we superimpose on the falling snow scene. As such, our method is a hybrid method. It combines geometry-based rendering (particle systems) with image-based rendering (spectral synthesis). Our method produces visually pleasing results at interactive rendering rates and is flexible enough to accommodate camera movement as well as changes in the direction and speed parameters of the spectral snow.

The remainder of our paper is organized as follows. In Section 2 we discuss related work. Sections 3 and 4 describe motion in the frequency domain and the dispersion relation for falling snow. In Section 5 we show how our spectral synthesis model is derived. In Section 6 we present experimental results. In Section 7 we present the theory and some results for dealing with a moving camera. Finally, we conclude and discuss some ideas for future work in Section 8.

2. Related Work

The idea of a combining image-based and geometry-based rendering is not new. For example, *Billboards* are a well-known technique for representing complex geometry, such as trees, with very few texture-mapped polygons. More sophisticated work, such as that of [MS95], [SLS*96], [SGwHS98], [AL99], or [WM02], replaces distant geometry with images. Our work is similar in that we replace a large number of discrete snow particles with a synthesized spectral snow in order to reproduce the effect of heavy snowfall in a large spatial volume. However, our approach not only reduces the rendering cost, but in our experimental system, the results looked better than simply increasing the number of particles.

For the case of a moving camera (Sec. 7), the synthesized spectral snow is related to view-dependent texturing [DTM96]. In traditional view-dependent texture mapping, the imagery changes based on the camera's position and orientation relative to a surface. In our case, the imagery changes based on the camera's orientation and motion relative to a volume of moving particles.

2.1. Spectral synthesis methods

Spectral synthesis methods have been used widely in computer graphics for 3D modeling of fractal-like objects. These methods grew out of pioneering work of [Man77] and [Vos88]. Examples of objects that have been rendered with spectral synthesis are terrains [FFC82, MKM89], ocean waves [MWM87, Sak93], static clouds [Gar85] or dynamic clouds [Sak93], fluids [Sta01], wind [SF92, SF93], fire and smoke [EMP*03].

A key property of objects rendered with spectral synthesis is a characteristic random multi-scale geometry. The visual appearance of the object is determined by the statistics of the geometry over scale, rather than by a deterministic smooth parametric model of the geometry. The object is obtained by summing up large numbers of sinusoidal functions which typically have random phase with respect to each other. The object is then rendered from this sum. Therefore,to model a given type of object, one must define an appropriate summation, i.e. the appropriate sinusoids and each sinusoid's contribution to the sum. One of the main contributions of our paper is an image-based method for generated the multi-scale motion texture properties of falling snow.

Spectral synthesis can be carried out in either the space-time domain [FFC82, Per85, Lew87, Lew89, MKM89] or the frequency domain [Sta01, Sak93, SF92, MWM87]. In the space-time domain, the object can be rendered procedurally at each image pixel and frame. This has the advantage that one needs only to render the visible points and at scales that are relevant for the viewing distance. One can also let the parameters of the model such as fractal dimension vary continuously across space.

The alternative is to render in the frequency domain. Here the disadvantage is that the parameters are specified globally, but the advantage is that one can use an inverse fast Fourier transform (IFFT) which is very fast. As the FFT can be computed on a GPU [MA03], one can potentially perform spectral synthesis of video in real time.

A simple way to bridge these two extremes of pure space-time vs. global Fourier transforms is to perform Fourier transforms locally within small image tiles. Though the frequency domain parameters are constant within each tile, they can vary from tile to tile. This is closely analogous to spectrogram methods used in classical speech analysis/synthesis [RJ93]. Using tiles in this way gives a local spatial control over the rendering parameters.

3. Motion in the frequency domain

Our spectral synthesis method for falling snow is reminiscent of the model for ocean waves introduced in [MWM87]. In that paper, a set of 2D waves is synthesized by summing 2D sinusoids such that the speed of each component sinusoid depends on spatial frequency in a manner dictated by a physics of ocean waves. The method we introduce for falling snow also sums up 2D translating sine waves, but in our case the waves represent the image motion of the snow at different depths. Before we derive our model, we present the necessary background.

In general, a 2D function such as an image that undergoes a *constant translation* over time yields a plane of power in the 3D spatiotemporal frequency domain[WA85]. If the image is translating with velocity (v_x, v_y) pixels per frame, then one can write:

$$I(x,y,t) \;=\; I(x - v_x\, t,\; y - v_y\, t, 0)\;.$$

Let (ω_x, ω_y) be the spatial frequencies in the x and y directions, and let ω_t be the temporal frequency. If one takes the 3D Fourier transform of the translating image $I(x,y,t)$, then one finds that all the power in the 3D frequency domain $(\omega_x, \omega_y, \omega_t)$ lies on the plane:

$$\omega_t \;=\; -v_x\omega_x - v_y\omega_y\;. \qquad (1)$$

This plane passes through the origin. We refer to it as the *motion plane* for velocity (v_x, v_y).

One way to understand the motion plane property of pure translation is as follows. When an image sequence is created

by translating a single image frame over time with velocity (v_x, v_y), then each of the 2D component sinusoids of the single image frame travels with this velocity as well. Thus, each component 2D sine wave produces a unique spatiotemporal frequency component in the translating image sequence. For each single frame component (ω_x, ω_y), there is a unique temporal frequency as governed by Eq. (1). For example, if the image velocity is purely in the y direction with speed s, i.e. $(v_x, v_y) = (0, s)$, then

$$\omega_t = -s\,\omega_y \tag{2}$$

For fixed s, higher spatial frequencies produce higher temporal frequencies. This is intuitively what one expects. The more cycles there are across a given pixel distance, the more temporal cycles there are (at a point) as those pixels translate with speed s.

Eq. (1) is merely a generalization of Eq. (2) in which the translation velocity is in an arbitrary image direction.

A few details on the geometry of the motion plane may help the reader's intuition. If one takes the intersection of the motion plane with the plane $\omega_t = 0$, one obtains a line:

$$v_x\omega_x + v_y\omega_y \;=\; 0\,.$$

The direction of the image velocity vector $(v_x,\ v_y)$ is perpendicular to this line, and the speed $v = \sqrt{v_x^2 + v_y^2}$ is the slope of the motion plane in this motion direction.

4. A dispersion relation for falling snow

Our model departs from the motion plane model by considering the effects of linear perspective. The image motion generated by falling snow is not a single translation but rather is a family of translations. Even when all snowflakes move with exactly the same *3D velocity*, snowflakes at different depths move with different *2D image velocities*.

We first address the case that the 3D velocity of the snowflakes is in a direction that is orthogonal to the camera's view vector. For example, all snow flakes might be moving in the y direction. In this case, the image speed of each snowflake is also in the y direction. But because of linear perspective, the speed of the snowflake will depend on its depth.

One way to capture the resulting range of image speeds that results would be to use a *set of* motion planes:

$$\{\ \omega_t = -s\cos\theta\ \omega_x - s\sin\theta\ \omega_y\ \ :\ s \in (s_{min}, s_{max}\ \} \tag{3}$$

where (s_{min}, s_{max}) is the range of image speeds, and $(\cos\theta, \sin\theta)$ is the fixed direction of image motion. Such a model was introduced in [LM03] for describing the 3D power spectrum of the images seen by an observer who is moving laterally relative to a cluttered 3D scene such as the woods. (Another example is the case of falling snow, of course.) In [LM03], the model of Eq. (3) was used for computer vision, rather than computer graphics. That is, the model was used for spectral analysis of video rather than for spectral synthesis.

The model of Eq. (3) does capture the *range* of speeds present in falling snow. However, it does not capture linear perspective effects of falling snow. In this paper, we extend the model of Eq. (3) to account for linear perspective and then apply the model to the spectral synthesis problem.

The main idea is to use multiple motion planes as in Eq. (3) but to restrict the set of frequencies contributed by each motion plane in a way that is consistent with linear perspective. The image of falling snow at depth d in the scene has two scaling properties associated with it, both of which arise from linear perspective:

First, *the closer a snowflake is to the camera, the faster the snowflake moves in the image.* Again we are assuming that all snowflakes are falling with roughly the same 3D velocity, and so the image speed s of a snowflake is is inversely proportional to depth d,

$$s \;\propto\; \frac{1}{d}\,.$$

Take the case of snowflake motion in the y direction. Substituting Eq. (2) we get:

$$d \;\propto\; \frac{1}{\omega_t/\omega_y} \tag{4}$$

Second, *the closer the snowflake is to the camera, the larger the snowflake appears in the image.* Larger image structure yields more power in lower spatial frequencies, i.e. smaller values of $\omega = \sqrt{\omega_x^2 + \omega_y^2}$. Thus, the depth d of a snowflake is proportional to the spatial frequencies to which the snowflake contributes:

$$d \propto \sqrt{\omega_x^2 + \omega_y^2} \tag{5}$$

Combining Eqns. (4) and (5) yields, for the case of motion in the y direction:

$$\omega_t/\omega_y \;\propto\; \frac{1}{\sqrt{\omega_x^2 + \omega_y^2}}$$

For a general motion direction θ, one obtains a more general *dispersion relation*:

$$\omega_t \;=\; C\frac{\cos\theta\ \omega_x + \sin\theta\ \omega_y}{\sqrt{\omega_x^2 + \omega_y^2}} \tag{6}$$

This dispersion relation is the basis for our spectral synthesis method.

The constant C is equal to the temporal frequency that corresponds to the "fundamental" spatial frequency, $(\omega_x, \omega_y) = (\cos\theta, \sin\theta)$. This fundamental is most easily understood in the case that $\theta \in \{0, \pm\frac{\pi}{2}, \pi\ \}$. In this case, C is the image speed of a sinusoidal component with one wavelength over the image domain.

The dispersion relation of Eq. (6) is a surface in the 3D frequency domain. Indeed it is a function, mapping spatial frequencies to temporal frequency. We refer to this as the *tent surface* because it has the appearance of a tent. A plot of the surface is shown in Figure 1.

The tent surface is not defined at $(\omega_x, \omega_y) = (0,0)$ since this is the dc component of the image sequence which obviously does not move. The behavior at very low spatial frequencies is governed by the following limits,

$$lim_{\,(\omega_x,\omega_y)\to(0,0)}\ \omega_t \;=\; \begin{cases} 1, & \cos\theta\,\omega_x + \sin\theta\,\omega_y > 0 \\ -1, & \cos\theta\,\omega_x + \sin\theta\,\omega_y < 0 \\ 0 & \cos\theta\,\omega_x + \sin\theta\,\omega_y = 0 \end{cases}$$

By inspection, one can also see that $\omega_t \in [-C, C]$ for all (ω_x, ω_y). This fact becomes important when we consider temporal aliasing and motion blur (see Sec. 5.5).

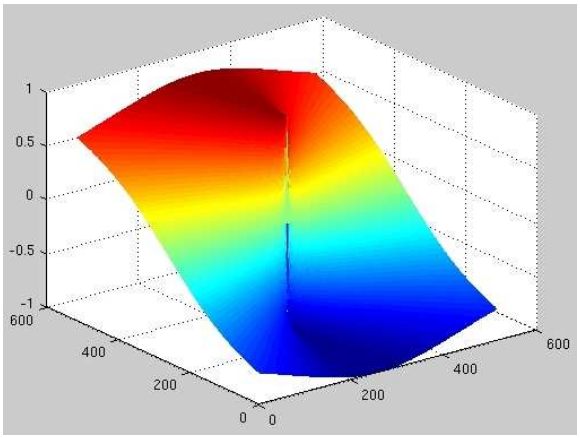

Figure 1: *Example of tent surface of Eq. (6).*

5. Details of Spectral Synthesis Method

In this section, we describe how to use the dispersion relation as the basis for a spectral synthesis method. Our method has two steps. First we use spectral synthesis to create a time-varying *opacity* function $\alpha(x,y,t)$ that represents the summed density of snowflakes that project to each pixel and frame. In terms of spectral synthesis, $\alpha(x,y,t)$ is the sum of moving 2D sine waves, as described by the tent surface of Eq. (6). This opacity function is rendered in image *tiles* which are then overlapped to span the image domain. Second, we use this opacity function to composite a white falling snow "layer" on top of a background image, video, or scene.

The next several subsections cover the following details: how the tent surface can be defined in the 3D and 2D domains; what amplitudes are chosen for each of the frequency components; how to address aliasing problems; and how to composite the snow over the background scene.

5.1. Discretization: 3D method

Consider a single $M \times M$ tile. We wish to synthesize the time varying opacity function for this tile over a sequence of T frames. One way to do so would be to construct a density function $\hat{\alpha}(\omega_x, \omega_y, \omega_t)$ in the $M \times M \times T$ frequency domain, such that the density is non-zero only on the tent surface, and then take the inverse FFT of this 3D density function, i.e.

$$\alpha(x,y,t) = \sum_{\omega_x=0}^{M}\sum_{\omega_y=0}^{M}\sum_{\omega_t=0}^{T} \hat{\alpha}(\omega_x,\omega_y,\omega_t)\, e^{i\frac{2\pi}{N}\omega_x x} e^{i\frac{2\pi}{N}\omega_y y} e^{i\frac{2\pi}{T}\omega_t t}$$

This yields the opacity function $\alpha(x,y,t)$ which is $M \times M \times T$ in space-time, and whose power spectrum by definition is the tent surface. This spectral synthesis method was used in [LZ03].

To construct $\hat{\alpha}(\omega_x, \omega_y, \omega_t)$, each cell in the frequency domain is initialized to 0. Any cell $(\omega_x, \omega_y, \omega_t)$ that overlaps the tent surface of Eq. (6) is then assigned a complex value with an amplitude varying between 0 to 1 (see Sec. 5.4) and a random phase varying between 0 to 2π. To ensure that the inverse Fourier transform $\alpha(x,y,t)$ of the function is real, one enforces a conjugacy constraint, namely that $\hat{\alpha}(\omega_x, \omega_y, \omega_t)$ is the complex conjugate of $\hat{\alpha}(N-\omega_x, N-\omega_y, T-\omega_t)$ [Bra65].

This 3D spectral synthesis method has two key limitations, however. First, it is slow. For an $M \times M = 128 \times 128$ image tile and $T = 64$, several MB are required for representing the 3D frequency volume and the computation of the IFFT takes several minutes on a Pentium 4. A second limitation is that the direction and range of speeds of the snow must be fixed over all T frames.

The 2D method which we describe next avoids both of these limitations.

5.2. Discretization: 2D method

Because the dispersion relation expresses ω_t as a function of ω_x and ω_y, we can substitute this relation and get rid of the ω_t variable in the IFFT above:

$$\alpha(x,y,t) = T \sum_{\omega_x=0}^{M}\sum_{\omega_y=0}^{M} \hat{\alpha}(\omega_x,\omega_y,t)\, e^{i\frac{2\pi}{N}\omega_x x} e^{i\frac{2\pi}{N}\omega_y y} e^{i\frac{2\pi}{T}\phi(\omega_x,\omega_y,t)}$$

where $\phi(\omega_x, \omega_y, t)$ is the *phase* which is a function of spatial frequency (ω_x, ω_y) and frame number t.

If C and θ are constant over time t, then $\phi(\omega_x, \omega_y, t)$ can be written as a product:

$$\phi(\omega_x,\omega_y,t) \;=\; C\,\frac{\cos\theta\,\omega_x + \sin\theta\,\omega_y}{\sqrt{\omega_x^2+\omega_y^2}}\;\phi_0(\omega_x,\omega_y)\;t$$

The ϕ_0 term is a random initialization of the phases, similar to what was described above for the 3D synthesis. In particular, these phases also must obey the conjugacy constraint to ensure that the IFFT is real for each frame.

In the more general case that C and θ vary with t, phase is updated from frame t to $t+1$ via:

$$\phi(\omega_x, \omega_y, t+1) := C(t) \frac{\cos\theta(t)\ \omega_x + \sin\theta(t)\ \omega_y}{\sqrt{\omega_x^2 + \omega_y^2}} \phi(\omega_x, \omega_y, t)$$

The 2D method just described reduces the spatiotemporal (3D) spectral synthesis problem to a spatial (2D) spectral synthesis problem, with the latter performed once per frame t. This eases the computational burden tremendously. We do not have to fit the $O(M^2T)$ floats – the size of the spectral domain – into main memory to compute the FFT. Instead, we need only fit $O(M^2)$ floats.

The more interesting advantage is that the various parameters of the motion can now vary with t:

- the fundamental speed C, *i.e.* $C(t)$
- the direction of the motion θ *i.e.* $\theta(t)$
- the opacity amplitudes $|\ \hat{\alpha}(\omega_x, \omega_y)\ |$ *i.e.* $\hat{\alpha}(\omega_x, \omega_y, t)$.

We will see examples of each of these time-varying parameters later in the paper.

5.3. Range of spatial frequencies

If all spatial frequencies (ω_x, ω_y) were to contribute to the tent surface, then the contributing wavelengths would vary from the size of the image tile M to the distance between pixels. We found that such extreme large or small wavelengths do not add to the visual impression of the snow. Presumably too large snowflakes don't work because they are so much larger than the particles, and too small wavelengths don't work because they correspond to snowflakes way off in the distance, and these move so slowly that their motion is not even noticeable.

We found that we could get the best appearance of the motion texture of falling snow when about three octaves of speeds were used – that is, a factor of 8 range.

For the spectral snow presented in the paper, we include power in the tent surface for three octaves of spatial frequencies, namely ω from $\frac{M}{32}$ to $\frac{M}{4}$ cycles per tile width, where $\frac{M}{2}$ is the spatial Nyquist frequency. Effectively, we are defining an annulus in (ω_x, ω_y) and assigning power only to spatial frequencies that lie in this annulus.

Our image tiles are of size $M \times M = 64 \times 64$. The Nyquist frequency is $\frac{M}{2} = 32$ cycles per tile and the annulus is the three octave range from 2 to 16 cycles per tile.

5.4. $1/\omega$ amplitudes

To make each of the image speeds of the spectral snow equally visible, we put a constant amount of power within each octave band of spatial frequencies [BF95]. We do so by assigning the amplitudes to have a $1/\omega$ noise:

$$|\ \hat{\alpha}(\omega_x, \omega_y)\ | \ = \ \frac{1}{\sqrt{\omega_x^2 + \omega_y^2}}$$

These amplitudes are defined once for each tile.

5.5. Temporal aliasing and motion blur

Temporal aliasing occurs for the tent surface when the temporal frequency ω_t is greater than the Nyquist frequency $\frac{T}{2}$, *i.e.*

$$|C \frac{\cos\theta\omega_x + \sin\theta\omega_y}{\sqrt{\omega_x^2 + \omega_y^2}}\ | \ > \frac{T}{2}$$

Aliasing causes high speeds in direction θ to appear as high speeds in direction $\theta + 180°$.

For C fixed, aliasing can be avoided by temporal blurring. Temporal blurring can be implemented in the 3D frequency domain by setting to zero the amplitudes of any spatial frequency components that obey the above inequality. For example, one could multiply $\hat{\alpha}(\omega_x, \omega_y)$ by a Gaussian weighting of the height of the tent surface ω_t, which is determined by (ω_x, ω_y). This weighting function drops to zero at the temporal Nyquist frequency.

The ease with which we can create motion blur is an advantage of the spectral synthesis method. With particle systems, motion blur often involves rendering the same particle multiple times, which is potentially costly.

5.6. Inverse FFT to obtain opacity

Once the function $\hat{\alpha}(\omega_x, \omega_y, t)$ has been computed for frame t, we can obtain the opacity function $\alpha(x, y, t)$ for frame t by taking the 2D inverse FFT.

To treat $\alpha(x, y, t)$ as opacity, we map it to the interval [0,1]. We do so by shifting the mean to 0.5 and reducing the standard deviation so that nearly all values lie in [0,1], clipping the outliers. We also take account of the fact that the human visual system is sensitive to logarithmic differences in intensity, rather than linear differences; after mapping to [0,1], we apply a non-linear transformation, namely we square the $\alpha(x, y, t)$ values. This compresses the opacity values to the lower part of the interval [0,1]. Thus, after the compositing step (described next), the variations in opacity are more visible.

5.7. Composite spectral snow with a background video

Finally, we composite the opacity with a background still image $I_{bg}(x, y)$ or dynamic scene $I_{bg}(x, y, t)$. We set the intensity of the foreground image sequence (the spectral snow layer) to $I_{snow} = 250$ since snow should appear white, and use the formula:

$$I(x, y, t) \ = \ I_{snow}\ \alpha(x, y, t) \ + \ (1 - \alpha(x, y, t))\ I_{bg}(x, y, t) \quad (7)$$

This is a variation of standard compositing [Bli94], in which the foreground intensity is now constant and the opacity $\alpha(x, y, t)$ varies with time.

Model	*Number of Polygons*	*Spectral Snow Resolution*	*Low Particle Count*	*High Particle Count*
Human Condition	6	512×512	2000	16,000
Son of Man	8	512×512	2000	16,000
Ventana	6	512 × 512	N/A	N/A
Flythrough	3836	1024×1024	30,000	150,000

Table 1: *Scene details. Each snow sequence was 30 frames.*

6. Results

6.1. Rendering times

The 2D method described above was implemented in C on a Pentium 4. For an image size of $N \times N = 512 \times 512$, spectral snow was generated at 4-5 frames per second, with a per-frame breakdown as follows: 40 *ms* for the phase update, 120 *ms* for the FFT, 30*ms* for the compositing, and 25 *ms* for writing to disk. Performing the computation on overlapping tiles that span the same image size does not significantly change these timing numbers. The numbers might be improved dramatically if we were optimize the code by taking advantage of a particular processor's floating point pipeline for the IFFT, or performing the computation on the GPU [MA03]. This is a topic for future work.

6.2. Example videos for single background images

We have submitted three demonstration scenes. All were implemented using DirectX 9 on a Windows XP PC with a Pentium 4, 2.4GHz processor and 1GB of RAM. In addition, the PC has an ATI Radeon 9800 Pro graphics card with 256 MB of texture memory. For all scenarios, the video textures are loaded as 8-bits-per-pixel luminance textures. Each spectral snow sequence is 30 frames, and we render these snow textures at 30 fps. See Table 1 for the parameters used. Our snow particle system used the CParticleSystem class implemented in the DirectX 9 PointSprites sample application. The video scenes submitted with this paper were rendered to disk frame-by-frame using a synthetic clock with a 1/30th of a second delay between frames. However, all of our demonstrations run at much higher interactive rates, as shown in Table 2.

Our first example `human_condition.avi` is based on a René Magritte painting of the same name, and was inspired by [HAA97]. We divided the painting into two textures: a foreground texture that corresponds to the interior scene (the room, ball, canvas, and easel) and a background texture that corresponds to the exterior scene (the sea and sky). The background texture and spectral snow are rendered orthographically onto a polygon at a far distance from the camera. The spectral snow, which mimics a constant downward snow fall, has a resolution of 512×512 pixels per frame, and is tiled across the polygon. Since each tile is toroidal, there are no seams across tile boundaries. Seamless tiling is advantageous since it allows us to use smaller textures.

Closer to the camera, we render a second polygon with only the foreground texture and no spectral snow. Snow particles are then dropped into the scene in the space between the two textured polygons. The spectral snow was generated to match the speed of the snow particles closest to the camera. The entire scene can be rendered in under 2 milliseconds. (See Table 2.)

In the accompanying video, we see that combining the spectral snow with the particle system looks better than either one alone. The particles provide visible individual snow flakes, but do not make the scene look full of snow. The spectral snow gives an atmospheric textural effect of heavy snow fall, but is lacking in individual snow particles and can suffer from a "shower door" effect. Together, the scene looks as if it has a heavy snowfall. Just as significantly, simply increasing the number of particles (from 2000 particles to 16000 particles) does not yield the same visual effect as that of combining the particle-based and image-based systems. We believe the hybrid system more closely mimics the visual effect of heavy snowfall. Furthermore, it does so with significantly faster rendering rates. In our timing tests, rendering rates slowed to an average of nearly 10 *ms* per frame with the increased number of particles.

The `son_of_man.avi`, also based on a picture by Magritte, was rendered in much the same way as the human condition demo.

The `ventana.avi` sequence demonstrates how the motion blur method described in Sec. 5.5 can be used to generate a motion texture that is reminiscent of falling rain. Here we used vertical motion direction and a high value of C, such that the only spatial frequency components (ω_x, ω_y) that contributed to the spectral sum were those in which $|\omega_y|$ was near zero, that is, only long wavelengths in the y direction.

All of these demos were quite easily created, showing that our method is a practical way to add convincing weather effects to 2D imagery.

7. Falling snow as seen by translating camera

We next consider the case of falling snow as seen by a moving camera. This case is more challenging since the directions and speeds of the spectral snow must be made to vary in a manner consistent with the particles, whose motions themselves can vary across the image.

We assume the falling snowflake particles have a constant 3D velocity, and so the velocity of the 3D snow in camera coordinates at any t is the difference of the snow's 3D velocity vector and the camera's 3D velocity vector. To render the spectral snow in camera coordinates, without further loss of generality we treat the 3D snow velocity as fixed and assume

the camera as moving with a resultant 3D velocity, which we denote (T_x, T_y, T_z).

We apply the equations of image motion developed in [LHP80] which describe the image velocity field seen by a camera moving relative to a rigid 3D scene (see also [TV98]). If the image plane is at depth $Z = f$, and $Z(x,y)$ is the depth of a 3D point visible at image position (x,y), then the image velocity at (x,y) is:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \frac{T_z}{Z(x,y)} \begin{bmatrix} x - x_T \\ y - y_T \end{bmatrix} \tag{8}$$

where the special image position

$$(x_T, y_T) = \frac{f}{T_z}(T_x, T_y)$$

is called the *focus of expansion* (FOE).

If the scene were a single fronto-parallel plane, then image velocity (v_x, v_y) would increases linearly with image distance from the FOE. This is the case shown in Fig. 2. The FOE, (x_T, y_T), is at the center, the second row from the top.

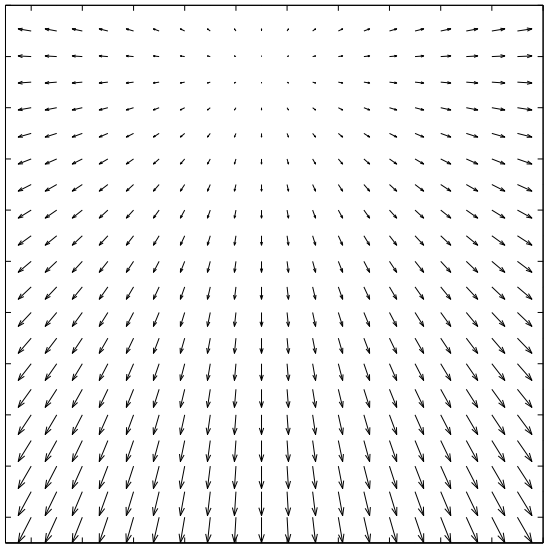

Figure 2: *Image velocity field seen by a camera moving forward and upwards. The depth map is a single fronto-parallel plane i.e. $Z(x,y)$ = constant.*

When the scene is 3D snow, there are many depth planes to consider. In each image tile, multiple velocity vectors are present, corresponding to different $Z(x,y)$ values of snowflakes that project to this tile. For any given range of depths snowflakes within a tile, the range of velocities in that tile scales linearly by the image distance to the FOE. Patches near the FOE have little or no motion, even though they may have a large range of depths.

To account for these spatially varying properties of the velocities, both between and within tiles, we rendered the spectral snow using a 1024×1024 image with overlapping small tiles of size $M \times M$ where $M = 64$. The overlap was 10 pixels, hence there were approximately 20×20 tiles. This is the sampling shown in Fig. 2. Spectral snow for neighboring tiles was blended linearly across the tile overlap region.

The direction θ_{ij} was chosen to be constant within each tile (i,j). This constant θ_{ij} was determined by the direction of (v_x, v_y) for the center point (x_i, y_j) of the tile, namely,

$$\theta_{ij} = \arctan\left(\frac{y_j - y_T}{x_j - x_T}\right)$$

The range of speeds within each tile was determined by the C constant which also varies from tile to tile, namely the range increases linearly with the image distance of the tile from the FOE:

$$C_{ij} = C_0 \sqrt{(x_j - x_T)^2 + (y_j - y_T)^2}\,.$$

The constant C_0 is chosen once for the whole image, such that the range of speeds at each position (x,y) in the spectral snow field roughly matches the range of image speeds of the snow particles.

7.1. Example video for moving camera

`Flythrough.avi` shows an example, which is based on a modified version of the DirectX 9 "Billboard" sample application. The spectral snow opacity function is mapped onto a polygon close to the near clipping plane. The video is not tiled; a single frame covers the entire polygon. Again, we see that the hybrid system yields better looking results than either particles or spectral snow alone. In the second video of this scene, `comparison.avi`, we can observe again that significantly increasing the number of particles (from 30,000 to 150,000) does not lead to the same visual effect as our hybrid approach, and it significantly decreases performance, causing frame rates to drop below the threshold for interactivity as we can see in Table 2. (We rendered the heavy snowfall off-line to achieve 30 frames per second in the video.)

7.2. Example of time-varying parameters

In the previous example, we described how the parameters C and θ of the dispersion relation can be varied from one image tile to the next. The final example shows how these parameters can also be varied over time. This would be necessary, for example, if the moving camera were to change translation direction, accelerate, or rotate (pan, tile, roll).

The video `FOEmoving.mpeg` shows spectral snow generated by simply varying the x_T coordinate of the FOE (x_T, y_T) over time.

Model	*Rendering Times Per Frame (ms)*				
	Basic Scene	*Low Particle Count*	*High Particle Count*	*Spectral Snow*	*Low Particles Plus Spectral Snow*
Human Condition	1.57	1.73	9.82	1.52	1.99
Son of Man	1.37	1.7	9.73	1.59	2.0
Flythrough	0.94	24.6	121.9	1.44	24.6

Table 2: *Rendering performance for various scenes. Rendered to an 800×600 window. Note that while the number of particles may significantly affect rendering times, performance is less affected by adding in the spectral snow.*

8. Conclusions and Future Work

We have presented a hybrid geometry- and image-based method for rendering falling snow. There are two main contributions of this work. First, we introduce an image-based spectral synthesis method for rendering falling snow which is based on the size/speed/depth relationship that results from linear perspective. This relationship defines multi-scale textural properties of the snow. We show how to synthesize this dynamic texture frame-by-frame using a 2D IFFT on image tiles. The motion parameters are constant within tiles, but can vary between tiles and from frame-to-frame. The method we describe is quite flexible, enabling us to simulate effects such as motion blur and rain.

The second contribution is to use the spectral snow as a way of "filling in" the dynamic texture properties of a particle system. We use a standard particle system to generate a large number of particles with discrete positions and velocities, and we choose the parameters of the spectral snow within each image tile to be consistent with motion of those particles within that tile. Because the spectral snow fills in the textural properties, far fewer particles are needed, significantly improving rendering rates. We also argue that spectral snow can do a better job of conveying the atmospheric/textural properties of the snow than does a large number of particles. Furthermore, the spectral snow can be easily incorporated into existing 3D systems with texture mapping.

In regards to future work, we hope to incorporate more motion flexibility. In each of our examples, the snowflakes were assumed to move with a single 3D velocity, as if there were a rigid body filling the atmosphere. In particular, the direction θ of the spectral snow motion was constant within each image tile. However, real falling snow is often subject to wind and other fluid-like atmospheric phenomena, causing 3D velocity to vary with position. Indeed, there has been great effort in developing rendering methods that simulate precisely these effects, as cited in Section 2.

Our method could be extended so that the direction and speed of the 3D snow can vary as a function of depth. For a given image tile, each circle of radius $\omega = \sqrt{\omega_x^2 + \omega_y^2}$ represents part of the 3D view volume – namely, the points lying at a depth that is proportional to ω, and that projects to the pixels in this tile. The average 3D velocity of snowflakes in this part of the view volume defines an image velocity (similar to Eq. 8) which can be used to determine the slope and direction of the motion plane which the spatial frequency circle of radius ω contributes to the tent. The 3D snowflake velocities in the view volume could be determined using known techniques [Sta01, SF92, SF93, Sta97]. Hence, it is again possible to integrate the spectral method with a particle method.

A second topic for future research is that the range of depths of the snowflakes may vary from tile to tile. (Currently the spectral method assumes a fixed three-octave range of depths over all tiles.) For example, in an outdoor scene, pixels covering the ground should have an opacity function whose contributing spatial frequencies are different than those for pixels covering the sky. One way to implement this effect would be to vary the octave bandwidth across tiles. Although the depth map of the background video may not be given, it could be provided by a computer vision technique [HZ00] or painted in by a user. Such image-based interaction indeed may be preferable for some users than methods that require the user to specify 3D models.

Acknowledgements

This research was supported by grants from NSERC and FCAR and by a generous donation from ATI Technologies.

References

[AL99] ALIAGA D. G., LASTRA A.: Automatic image placement to provide a guaranteed frame rate. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 307–316. 2

[BF95] BRADY N., FIELD D. J.: What's constant in contrast constancy? the effects of scaling on the perceived contrast of bandpass patterns. *Vision Research 35*, 6 (1995), 739–756. 5

[Bli94] BLINN J. F.: Compositing, part I: Theory. *IEEE*

Computer Graphics and Applications 14, 5 (September 1994), 83 – 87. 5

[Bra65] BRACEWELL R. N.: *The Fourier Transform and Its Applications*. McGraw-Hill, NY, 1965. 4

[DTM96] DEBEVEC P. E., TAYLOR C. J., MALIK J.: Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM Press, pp. 11–20. 2

[EMP*03] EBERT D. S., MUSGRAVE F. K., PEACHEY D., PERLIN K., WORLEY S.: *Texturing and Modeling – A Procedural Approach*. Morgan Kaufmann, San Francisco, CA 94104-3205, 2003. 1, 2

[Fea00] FEARING P.: Computer modelling of fallen snow. *SIGGRAPH Proceedings* (2000), 37–46. 1

[FFC82] FOURNIER A., FUSSELL D., CARPENTER L.: Computer rendering of stochastic models. *Communications of the ACM 25*, 6 (1982), 371–384. 2

[Gar85] GARDNER G.: Visual simulation of clouds. *Computer Graphics 19*, 3 (1985), 297–304. 2

[HAA97] HORRY Y., ANJYO K.-I., ARAI K.: Tour into the picture: using a spidery mesh interface to make animation from a single image. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 225–232. 6

[HZ00] HARTLEY R., ZISSERMAN A.: *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000. 8

[Lew87] LEWIS J. P.: Generalized stochastic subdivision. *ACM Transactions on Graphics 6* (1987), 167 – 190. 2

[Lew89] LEWIS J. P.: Algorithms for solid noise synthesis. *SIGGRAPH Proceedings* (1989), 263 – 270. 2

[LHP80] LONGUET-HIGGINS H., PRAZDNY K.: The interpretation of a moving retinal image. *Proceedings of the Royal Society of London B B-208* (1980), 385–397. 7

[LM03] LANGER M. S., MANN R.: Optical snow. *International Journal of Computer Vision 55*, 1 (2003), 55–71. 3

[LZ03] LANGER M. S., ZHANG L.: Rendering falling snow using an inverse fourier transform. In *SIGGRAPH Technical Sketch (unpublished)* (2003). 4

[MA03] MORELAND K., ANGEL E.: The fft on a gpu. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003 Proceedings* (July 2003), Doggett M., Heidrich W., Mark W.,, Schilling A., (Eds.), pp. 112–119. 2, 6

[Man77] MANDELBROT B. B.: *Fractals: Form, Chance, and Dimension*. Freeman, San Francisco, 1977. 2

[MKM89] MUSGRAVE F. K., KOLB C. E., MACE R. S.: The synthesis and rendering of eroded fractal terrains. *Computer Graphics 23*, 3 (July 1989), 41 – 50. 2

[MS95] MACIEL P. W. C., SHIRLEY P.: Visual navigation of large environments using textured clusters. In *Symposium on Interactive 3D Graphics* (1995), pp. 95–102, 211. 2

[MWM87] MASTIN G. A., WATTERBERG P. A., MAREDA J. F.: Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Applications 7*, 3 (March 1987), 16 – 23. 2

[NIDN97] NISHITA T., IWASAKI H., DOBASHI Y., NAKAMAE E.: A modeling and rendering method for snow by using metaballs. *Computer Graphics Forum 16*, 3 (1997), 357–364. 1

[Per85] PERLIN K.: An image synthesizer. In *SIGGRAPH Proceedings* (July 1985), pp. 287–296. 2

[Ree83] REEVES W. T.: Particle system — a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics 2* (1983), 91 – 108. 1

[RJ93] RABINER L., JUANG B.-H.: *Fundamentals of speech recognition*. Prentice-Hall, Inc., 1993. 2

[Sak93] SAKAS G.: Modeling and animating turbulent gaseous phenomena using spectral synthesis. *The Visual Computer 9* (1993), 200 – 212. 2

[SF92] SHINYA M., FOURNIER A.: Stochastic motion - motion under the influence of wind. In *Eurographics '92 Proceedings* (September 1992), pp. 119 – 128. 1, 2, 8

[SF93] STAM J., FIUME E.: Turbulent wind fields for gaseous phenomena. *SIGGRAPH Proceedings* (1993), 369–376. 1, 2, 8

[SGwHS98] SHADE J., GORTLER S., WEI HE L., SZELISKI R.: Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), ACM Press, pp. 231–242. 2

[Sim90] SIMS K.: Particle animation and rendering using data parallel computation. In *SIGGRAPH Proceedings* (August 1990). Dallas, TX. 1

[SLS*96] SHADE J., LISCHINSKI D., SALESIN D. H., DEROSE T., SNYDER J.: Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM Press, pp. 75–82. 2

[SOH99] SUMNER R., O'BRIEN J. F., HODGINS J. K.: Animating sand, mud and snow. *Computer Graphics Forum 18*, 1 (1999), 17–26. 1

[Sta97] STAM J.: Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Computer Graphics Forum* (1997). 8

[Sta01] STAM J.: A simple fluid solver based on the FFT.

Journal of graphics tools 6, 2 (2001), 43 – 52. 1, 2, 8

[TV98] TRUCCO E., VERRI A.: *Introductory Techniques for 3-D Computer Vision*. Prentice-Hall, 1998. 7

[Vos88] VOSS R. F.: *The Science of Fractal Images*. Springer-Verlag, New York Berlin Heidelberg, 1988, ch. Fractals in nature: From characterization to simulation. 2

[WA85] WATSON A. B., AHUMADA A. J.: Model of human visual motion sensing. *Journal of the Optical Society of America A 2* (1985), 322 – 342. 2

[WM02] WILSON A. T., MANOCHA D.: *Spatially encoded image-space simplifi cations for interactive walkthrough*. PhD thesis, 2002. 2