Rendering Falling Rain and Snow

Niniane Wang*
Microsoft Corporation (now at Google Inc.)

Bretton Wade†
Microsoft Corporation

1 Modeling Precipitation

Realistic-looking rain and snow greatly enhance scenes of outdoor reality, with applications including computer games and motion pictures. We present a novel technique for realistically and efficiently rendering precipitation in scenes with moving camera positions. We map textures onto a double cone, and translate and elongate them using hardware texture transforms.

Most previous work modeled precipitation as a particle system [2], which produced realistic movement. We target real-time applications that require fast performance, which is often difficult with a particle system, especially for scenes of heavy precipitation. Inverse Fourier transforms [1] have been used to render falling snow over a static background image with real-time performance. Our technique addresses the additional challenge of rendering precipitation with a moving camera position. We also offer more fine-grained control over motion factors and drop appearance.

We model precipitation with four artist-generated textures, mapped onto a double cone centered about the camera, as shown in Fig. 1. We tile each texture to cover the cone, and we use a hardware vertex shader to blend all four together. We (1) tilt the cones to adjust for camera movement, (2) elongate the textures to simulate motion blur, and (3) scroll the textures to simulate gravity.

We scale down each of the four successive textures and scroll it more slowly, creating drops that are smaller and move slower to simulate depth with parallax. We increase the transparency of points closer to the cone endpoints to avoid artifacts, and ensure the cone is tall enough so that the slope of its edges is not noticeable.

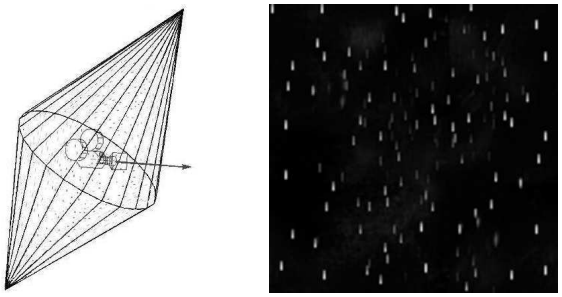

Figure 1: Left: Double cone. Right: Rain texture.

Fig. 2 shows a screenshot of falling snow. Animation sequences are included in the accompanying video.

As the camera moves, we pivot the double cone so that precipitation appears to fall toward the camera. We do this by setting the cone's world matrix to look along the precipitation travel vector Vec_{precip}, described in equation 1. Vel_{camera} is the camera velocity, C_f is a damping constant to limit the tilting of the cone, $Vel_{gravity}$ is the velocity of the precipitation due to gravity, and t_{delta} is the time elapsed since the last frame.

$$Vec_{precip} = \left(C_f * Vel_{camera} + Vel_{gravity}\right) * t_{delta} \tag{1}$$

We elongate drops to simulate motion blur. We compute the elongation factor E, described in equation 2, based on Vec_{precip},

Figure 2: A scene of snow in Seattle – a rare sight.

S_f, a pre-computed scale factor to scale down successive textures so they appear further away, C_{pixel}, a conversion factor specifying the real-world space equivalent for each pixel in the texture, and S_{streak}, a pre-computed scale factor that we use to give light rain longer streaks than heavy rain, for example.

$$E = \frac{S_{streak}}{S_f * \left(C_{pixel} * S_f * Vec_{precip} + S_{streak}\right)} \tag{2}$$

We use translation to make the precipitation fall with gravity. We scroll the texture each frame by D_{precip}, described in equation 3,

$$D_{precip} = S_{streak} * t_{delta} / t_{const} \tag{3}$$

where t_{const} is a fixed frametime, e.g. $\frac{1}{30}$ seconds, to make the precipitation fall at a constant rate regardless of framerate. D_{precip} and the elongation E are combined in the texture transform matrix. D_{precip} is represented in pixel space and converted to real-world space by E, which also slows down scrolling in successive textures.

Our system looks most realistic for rain and heavy snow. Light snow should jitter as it falls, which is better simulated by a particle system.

We experience a visual artifact when the framerate stutters drastically, in that the scrolling of the precipitation texture also stutters.

Our system allows fine-grained control over the outcome. We wrote a tool that allows tweaking of visual parameters, including S_f, S_{streak}, and C_f. Adjusting these values and the textures yields more control than was typically afforded in previous approaches.

Our technique has the same performance overhead for heavy precipitation as light precipitation, unlike a particle system. It ships with Microsoft Flight Simulator 2004: A Century of Flight, which maintains framerates of 15 to 60 fps across a range of consumer PCs. Our technique causes no noticeable degradation in framerate.

References

[1] M Langer and Q Zhang. Rendering falling snow using an inverse fourier transform. In *ACM SIGGRAPH 2003 Technical Sketches Program*, Computer Graphics Proceedings, Annual Conference Series. ACM, 2003.

[2] W T Reeves. Particle systems – a technique for modeling a class of fuzzy objects. In *Proc. of SIGGRAPH '83*, Detroit, Michigan. ACM, 1983.

*e-mail:niniane@ofb.net

†e-mail:brettonw@microsoft.com