Real-Time Rendering of Realistic Rain

Lifeng Wang, Zhouchen Lin, Tian Fang, Xu Yang, Xuan Yu, and Sing Bing Kang[1]

July 2006

Technical Report
MSR-TR-2006-102

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

[1]A one-page summary appeared as a technical sketch at SIGGRAPH, August 2006. Tian Fang is with South China University of Technology, P.R. China. Xu Yang is with Nankai University. Xuan Yu is with Shanghai Jiao Tong University, P.R. China. Tian Fang, Xu Yang, and Xuan Yu were visiting students to Microsoft Research, Asia, when this work was done.

Abstract

In this report, we propose a new technique to render realistic rain in real-time using a commodity graphics accelerator. Our technique consists of two parts: off-line image analysis of rain videos, and real-time particle-based synthesis of rain. Videos of real rain are analyzed to extract the rain mattes; random samples of these rain stroke mattes are then used for online synthesis. We incorporate pre-computed radiance transfer (PRT) in our particle system to take into account the scene radiance. We show that there is a closed-form solution to the transfer function of a raindrop, making its evaluation highly efficient. Our approach achieves high realism with low computation cost and a small memory footprint, making it very practical. We demonstrate our technique on a variety of scenarios, from synthetic 3D scenes to real videos.

1 Introduction

Weather simulation is an important effect that adds to the realism of a rendered scene. The ability to simulate weather effectively would be exceedingly useful for applications such as film post-production, games, simulators, and virtual reality. A lot of work has been done on simulating rain, fog, cloud, and snow.

Among all weather phenomena, rain is probably the most common. Interestingly, Garg and Nayar [5] analytically showed that with the proper settings of the camera and under certain rain conditions, rain can actually be made clearly visible or almost imperceptible in the video. In filming, it is possible that a scene with rain may be inadvertently captured with unintended results. In such a case, the option of adding realistic synthetic rain quickly in post-production would be an attractive and cost-effective one.

There has been a fair amount of work on rain simulation and synthesis. Yoshihiko et al. [17] combined a particle system with a physical motion model of the rain (with no GPU acceleration). Feng et al. [3] proposed a real-time GPU accelerated rain rendering algorithm that incorporated collision detection. Their system was also based on the particle system. However, rain synthesized by both these systems did not consider the effect of environment lighting.

Another approach for simulating rain is through direct physical simulation. This approach provides more flexibility in controlling rain, e.g., being able to freeze time and render a magnified raindrop. ATI's ToyShop demo [2] is an interesting example of this approach. While the result looks stunning, the rain does look repetitious, as it is based on a manually designed rain mask [1]. In addition, the realism came at a cost: it required 300 unique shaders dedicated to rain *alone* [1],

and the powerful Radeon® X1800 graphics card is required for real-time rendering. Furthermore, the influence of the environment on the rain is not too evident (the exception was the lightning effect on rain). In comparison, our technique extracts various shapes of rain from videos at low cost. We also incorporate pre-computed radiance transfer (PRT) to make the appearance of rain change with the environment map in real-time. As PRT uses only the low frequency spherical harmonics (SH) coefficients of the environment map and the transfer vectors of raindrops, the memory footprint is small. The memory requirement consists of environment maps (64 bytes each) and raindrop transfer vectors (6 MB, due to 16 SH coefficients per viewing direction times 98,304 viewing directions).

Starik and Werman [14] may have been the first to investigate image-based rain. After extracting rain from a video by median filtering over time, they applied the extracted rain to a different background or restored it to the median-filtered (derained) video. To make the rain more realistic, they desaturate the background image by averaging it with its luminance. They also blur the background image to account for the lens effect of the raindrops. Because the rendering is purely image-based, it has the appearance of a sheet of translucent texture. Their technique seemed to be limited to static scenes with fixed viewpoints, and is unable to physically account for scene radiance.

Garg and Nayar [4] proposed an approach to detect and then remove rain from videos. They find rain streaks by detecting instant changes in intensity and checking for directionality. However, they did not use the extracted rain for synthesis.

Our approach is a hybrid approach—it transfers details from video of real rain to a particle-based system. It combines the realism from images with the real-time performance of graphics techniques (specifically using pre-computed radiance transfer, or PRT). The decomposed rain strokes, modeled as particles, are uniformly distributed in the space with controllable velocity. Their colors are determined by PRT; our version of the transfer function of a raindrop has a closed-form solution for highly efficient evaluation. Our approach is cost-effective, yet capable of real-time realistic rain rendering.

2 Extracting Rain Strokes from a Static Scene

Although rain strokes can be manually designed, the process is tedious, and it is not guaranteed that the resulting simulated strokes will appear natural or sufficiently varied. As a result, we chose to extract rain strokes from actual footage involving real static scenes. As pointed out in [14] and [4], if the video was not captured using a high-speed camcorder, the rain in successive frames are

practically independent. Using this reasoning, Starik and Werman [14] and Garg and Nayar [4] simply used a median filter to extract or detect rain.

Unfortunately, the median filter is applied pixel by pixel independently. It does not take into account two relevant factors: rain is globally directional and a typical scene has a mostly continuous color distribution. As a result, it is less resilient to image noise. We incorporate these two factors in our rain extraction formulation as a nonlinear least-squares problem:

$$\begin{aligned}(\mathbf{S},\alpha) = \arg\min_{\mathbf{S},\alpha} \sum_{i,j,k} \left(I^k_{i,j} - (1-\alpha^k_i)S_{i,j} - \alpha^k_i C_j\right)^2 \\ +\lambda \sum_{i,j} ||\nabla S_{i,j}||^2 + \mu \sum_{i,k} \left(\nabla \alpha^k_i \cdot \mathbf{D}_0\right)^2,\end{aligned} \tag{1}$$

where indices i, j, and k are used for referencing pixels, color channels, and frames, respectively. I is the observed color, α is the fractional blending weight due to rain, S is the color of the static background, (C_R, C_G, C_B) is the "color" of the rain (due to the environment), $\mathbf{D}_0$ is the principal rain direction, and λ and μ are weights. The first term in (1) assumes the observed color is a linear blend of the colors of the scene and rain. The second term encourages smoothness in the scene color distribution, while the third favors smoothness of the alpha distribution along the rain direction $\mathbf{D}_0$. An Expectation-Maximization (EM) algorithm is applied to solve α and $\mathbf{S}$ alternately; the median filtered version of $\mathbf{S}$ is used to initialize $\mathbf{S}$. $\mathbf{D}_0$ is initialized as the vertical direction. It is refined immediately after estimating α by detecting the most salient line in the frequency spectrum of α that passes through the origin.

$\mathbf{S}$ and α can be solved as long as the video has more than three frames. In our work, we use 20 frames from a video of a typical rain scene in order to generate rain stroke samples to choose from, and set $\lambda = 1/20$ and $\mu = 1/4$ in (1). Once the rain direction $\mathbf{D}_0$ is known, the rain strokes are rotated to make them practically vertical. To reduce the artifacts caused by not fully modelling the rain (e.g., inter-reflection effects), we apply morphological erosion and dilation followed by vertical blurring. In practice, these operations are sufficient to produce plausible shapes of rain strokes. The best-looking individual rain strokes can be easily cut out as samples (top left corner of Figure 1).

3 Overview of Rendering System

In order to be able to exercise more control over the rain and enable the rain to appear natural with the scene, we chose to use a particle system for rendering, rather than using a single layer of rain [14]. Figure 1 shows the outline of the rendering process.

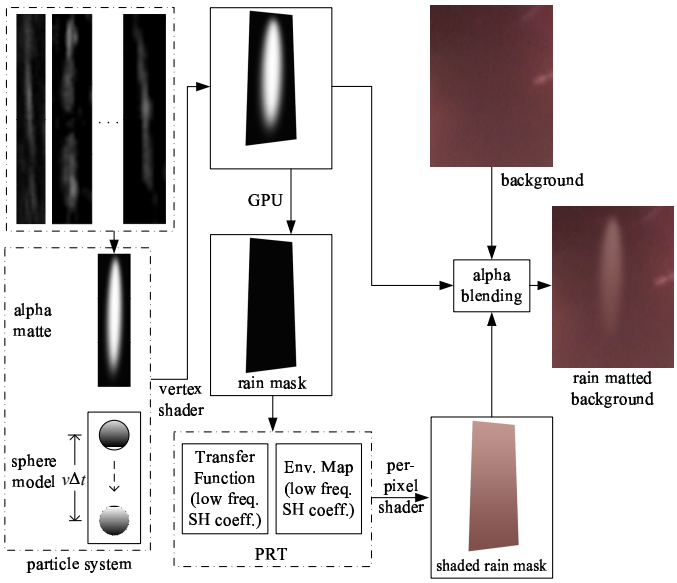
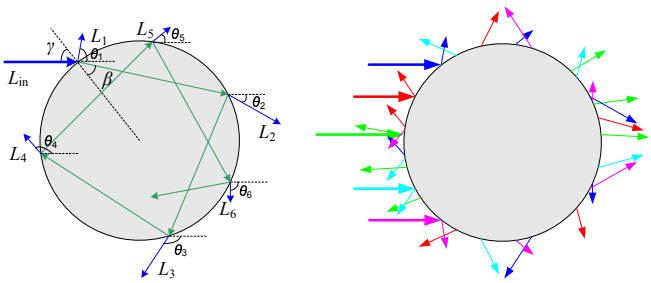


Figure 1: *Rain rendering process. Each particle consists of a rain stroke matte that is randomly chosen from the rain stroke samples, a sphere model, and attributes such as velocity. The vertex shaders compute the 3D position and orientation of the alpha matte and the GPU determines which pixel on the screen to shade. Our PRT based per-pixel shader then computes the color of each pixel on the rain stroke. Finally, the rain stroke and the background are blended using the rain alpha matte. "SH" refers to "spherical harmonics."*

Each particle has a sphere model with the refractive index of water, a random rain stroke matte, and other physical attributes such as position and velocity. The rain color is computed on the fly. In contrast, the particles of traditional particle systems [3, 17] only have the sphere model of the raindrop and other physical attributes. The rain color in their systems is prescribed, and the alpha value is computed from the portion of a raindrop in a pixel.

The particles are assumed to be uniformly distributed in space; we model the position as a uniform random variable. The length of the matte shape is estimated based on the particle's current velocity and the exposure time of the virtual camera. The matte shape is achieved by shrinking or stretching its original shape accordingly through interpolation. The diameter of the sphere that models the raindrop corresponds to the width of the rain matte after being mapped to the scene coordinate frame. We show that there is a closed-form solution to the transfer function required by PRT, which makes its evaluation highly efficient.

Our rendering technique is purely GPU based. (It requires pixel shader 2.0 and vertex shader 1.1, which are supported by most commercially available graphics cards.) After the 3D position and orientation of each particle have been computed by a vertex shader, the next step is to color



(a) (b)

Figure 2: *Sphere model for raindrop. (a) The reflection and refraction of light after hitting a raindrop. L_{in} is the incident light. It will be scattered by the raindrop toward different directions $L_1, L_2, \cdots$. (b) For incoming light rays (thick arrowed lines pointing towards sphere), the outgoing rays can then be computed (thin arrowed lines with corresponding colors pointing away from sphere). Five different incoming rays are shown here. Due to the reversibility of light, the transfer function of the raindrop is identical to the outgoing light intensity distribution function.*

the particle. The GPU first determines which part of the screen is to be shaded. A per-pixel PRT shader then computes the intensity distribution of the particle. Details of how the PRT shader works are deferred to Section 4. The environment map is used to compute the intensity and color of each pixel of a particle. By comparison, the ATI demo [2] used a normal map to compute the reflection/refraction of scene light (not the whole environment map) that hits the raindrop [1], which is more expensive than our PRT based method, and the nearest rain strokes appeared to be rendered using a sheet of rain mask.

Finally, the GPU blends the appearance of the particle with the background using the computed intensity and alpha distributions. Note that if rough geometry of the scene is known, the particle will be automatically clipped by the rendering engine at the scene surface (defined using a mesh of a synthetic scene or the depth map of a real video). This gives a volumetric appearance of rain.

4 Precomputed Radiance Transfer (PRT)

We developed a real-time PRT [13] engine that is capable of computing the color and intensity of each point on a raindrop in the scene. We assume that there is no inter-reflection among the raindrops, so that every raindrop is affected by the environment map only. The PRT consists of two parts: the environment map and the transfer function of the sphere model.

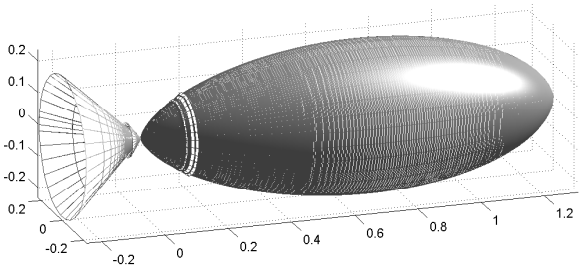


Figure 3: *The 3D shape of the transfer function of the raindrop when the viewing direction is* $(0, 0)$*. Direction* $(0, 0)$ *is from left to right along the major axis of the shape. The singular point* $(0, 0, 0)$ *corresponds to the raindrop location.*

4.1 Environment Map Generation

A video of a real scene alone typically does not provide a complete environment map. We merely filled in the missing parts using the nearest available colors. In all our real video examples, the horizontal field of view is approximately 36°. For a panning camera, we stitch the video into a panorama, which provides wider coverage for the environment map. For a looming camera, we stitch a panorama every five frames in order to produce view dependent environment maps.

The pixel values of the scene, particularly the light sources, are of low dynamic range and may not cause noticeable effects on the rain. Because of this, we replace the saturated values with the radiance of the light sources measured when we captured the video.

4.2 Transfer Function of Sphere Model

To use PRT, we have to compute the transfer function for each raindrop, i.e., the function that weights the environment map to produce the intensity of light reflected toward the viewing direction. Unfortunately, direct computation is difficult. Thanks to the reversibility of light, we may instead compute the outgoing light intensity distribution function by tracing the light in the viewing direction to find out how the light is scattered toward various directions (Figure 2). This function must then be identical to the transfer function.

To simplify the derivation, we model a raindrop as a sphere. (Its size is inconsequential since it is typically much smaller than the scene.) Using principles of basic optics (Appendix I), we can derive the closed-form solution for the outgoing light distribution function $D(\theta, \phi)$ for the spherical raindrop model (Appendices II and III). θ and ϕ are the angles associated with the altitude and azimuth, respectively, with $(0, 0)$ aligning with the z-axis (see Figure 2(a) for visualization of

θ). Because light is invertible, the transfer function $F(\theta, \phi)$ must be identical to the outgoing light intensity distribution function $D(\theta, \phi)$. Although there have been complex theories of light scattering by small particles (e.g. [15]), our formula is simple enough for fast computation while retains the main features of light scattering.

The transfer function $F(\theta, \phi)$, shown in Figure 3, is rotationally symmetric with respect to the azimuth angle ϕ. The part of the environment map corresponding to the viewing direction contributes the most to the radiance of the raindrop. There are some bumps in the transfer function, which are due to the discontinuity of light reflection when the incident angle of light reaches some critical angles. Note that parts of the environment map closer to the camera do contribute to the radiance of the raindrop as well. Since we model the raindrop as a sphere, the transfer function ((10) in Appendix III) differs only by a rotation for viewing directions other than $(0, 0)$.

Since the transfer function is closed-form, the expensive ray tracing associated with its computation is avoided. Moreover, as the transfer function in a different viewing direction differs only by a rotation, it is (in theory) adequate to just store the 1D transfer vector $\mathbf{f}_{00}$ for the viewing direction $(0, 0)$ and the transfer vectors for the remaining viewing directions can be obtained by applying a block diagonal matrix to $\mathbf{f}_{00}$ [6]. This would then result in an extremely small footprint (64 bytes). However, in practice, the rotation operation significantly adds to the computational load. As a result, we precompute the transfer vectors for all (98,304) viewing directions.

4.3 Rendering with PRT

When applying PRT, both the discretized transfer function and the environment map are projected to Spherical Harmonic (SH) bases; only the low frequency coefficients are stored. We store the first 16 SH coefficients for both the transfer function and the environment map. When shading a pixel in the rain stroke matte, the color C of the pixel is computed as the dot product of the raindrop transfer vector in the current viewing direction and the SH coefficient vector of the environment map.

Since we only keep the low frequency SH coefficients, the transfer function used actually contains just its low frequency part. In particular, the value of transfer function in the viewing direction is reduced the most (Figure 3). To compensate for this attenuation, we add a portion of the background color B (which is exactly the environment map in the viewing direction) to the rain color in the following manner:

$$C' = C + cB, \tag{2}$$

with c set to 0.7 in our implementation. The GPU then blends the corrected rain color with the

background, producing the final rain matted scene:

$$\tilde{C} = \alpha C' + (1 - \alpha)B,$$

where α is the alpha value of a pixel in the rain stroke matte. Using PRT, the effect of the environment map on the rain can be realistically achieved, as shown in Figures 4(b) and (c).

5 Additional Considerations

The basic algorithms described earlier allow us to render realistic-looking rain in real-time. However, we can significantly enhance realism by taking into account the following factors:

Rough Geometry. It is helpful to know the rough geometry of the scene to be able to assign the density of rain at the appropriate places. For a real scene and arbitrary camera motion, structure from motion techniques can be applied to extract scene depth. However, when there is no motion parallax (for a panning camera or flat scene), approximate depth would have to be assigned manually in general. Certain interactive techniques, such as the tour-into-the-picture approach proposed by Horry et al. [8], may be used to speed up the depth assignment process.

Scene Color. We use the simple darkening algorithm proposed by Starik and Werman [14] to mimic the grayish effect of a raining day. The amount of darkening can be interactively adjusted. The histogram based color transfer algorithm described in [12] can also achieve this goal.

Scene Blur. The refraction by the raindrops causes distortion of the background. The analysis of Narasimhan and Nayar [11] showed that the glow around light sources and the shape of the atmospheric point spread function (due to multiple scattering) are dependent on the sizes of the particles in the air. We approximate this effect by Gaussian blurring the video with a reasonable variance.

Rain Fog. Garg et al. [5] analytically showed that rain strokes are visible in the video only when they are close enough to the camera. When the rain strokes are far away, it has the appearance of fog. Therefore, to add realism to the rain scene, we also add homogenous fog. The alpha matte of the fog is exponentially decreasing with respect to depth d [18, 10], i.e., $\alpha_{\text{fog}} = 1 - \exp(-\varepsilon d)$, where ε is the fog density. We used values of ε between 1.35 and 1.8 in our experiments.

Rain Splatter. The shape of each rain splatter used in our experiments was designed by an artist. The size and number of the spatters linearly increases with the magnitude and density of the rain strokes. The splatters are uniformly distributed on the ground and alpha blended with the background.

(a) (b) (c)

Figure 4: *Adding synthetic rain to video of the Street scene. (a) An original frame of resolution* 720×480*, (b) result of adding rain, (c) close-up of the boxed area in (b). Notice how the synthetic rain was affected by the background. Here, 80,000 rain strokes (all visible to the virtual camera) were rendered at 78 fps on our Intel Pentium® 4 3.20GHz PC (with an nVIDIA® GeForce 7800 GT graphics card). The memory footprint dedicated to computing the lighting effect on the rain is only about 6 MB for the 33 second video.*

6 Results

The reviewer is strongly encouraged to check out our video submission, as the dynamic nature and realism of our synthesized rain are better observed through video. Due to size limit, we only present part of our demos in the submitted video. More demos can be found at: *http://research.microsoft.com/~zhoulin/Rain.htm*.

Our rain rendering results for a variety of scenes are shown in Figures 4, 5, 6, and 7. They demonstrate the realistic effect of the environment on the synthetic rain. Even though the same rain stroke samples are used in all the scenes, their final appearance looks significantly different.

User-controlled magnitude of rain. The user can easily control the magnitude of rain, as shown in Figures 5(d)-(f). This feature has interesting possibilities as a movie post-production tool. We can also add rain to synthetic scenes, as demonstrated in Figure 6. The speed of rain can also be controlled, but it is difficult to illustrate with figures.

Comparison with other traditional techniques. In addition, we compare our results with those using a traditional particle system and by ray tracing (Figure 7). Our result (Figure 7(a)) is very close to that obtained using ray tracing (Figure 7(b)); however, our rendering speed is hundreds of times faster. The rain synthesized using a traditional particle system (Figure 7(c)) is much less sensitive to environment lighting. Figure 7(d) shows why a variety of strokes is necessary. Using hand-drawn simple rain stroke shape resulted in less natural looking rain (compared to Figure 7(a)).

Implementational details and statistics. Table 1 shows the data size for the various scenes. For

Table 1: *Memory footprints for different scenes.*

Scene	Video Length (seconds)	SH Coeff. Footprint (bytes/panorama)	Total (bytes)
Street	33	64	12,672
Yard	16	64	64
Gateway	24	64	64
Roof	19	64	64

Table 2: *Memory cost dedicated to lighting effect of rain. We used 64 rain samples, and the system stored transfer vectors corresponding to 98,304 viewing directions.*

	Rain Stroke Mattes	Raindrop Transfer Vectors
Unit Footprint (bytes)	1,024	64
Number of Entities	64	98,304
Total Footprint (MB)	0.0625	6

the Street scene, a panorama was stitched every 5 frames to create view dependent panoramas. For the other real scenes (with panning camera), only one panorama was stitched. For each panorama, only 16 low frequency spherical harmonics (SH) coefficients were stored.

Table 2 lists the memory footprint dedicated to the lighting effect of the rain. The memory costs for the rain stroke samples and the raindrop transfer vectors are both quite small. Although there are some other memory costs, such as the video sequence and the depth map, they are unrelated to the lighting effect of the rain.

We used a PC with Intel Pentium® 4 3.2GHz CPU, 1GB RAM, and an nVIDIA® GeForce 7800 GT graphics card. Table 3 lists the average frame rates with different numbers of rain strokes. Even with 80,000 rain strokes for each frame (all the rain strokes are visible to the virtual camera), the frame rate we obtained was 77 fps.

7 Discussion

Currently our rain splatters are crafted by hand. While it may be possible to extract them from video, the process is substantially more complicated due to accompanying effects such as ripples.

Table 3: *Average frame rate vs. number of rain strokes rendered to* each frame *of scene. All the rain strokes are visible to the virtual camera.*

# of Strokes ($\times 10,000$)	1	2	3	4	5	6	7	8
FPS	790	328	223	162	130	104	91	77

As a result, we cannot directly apply our technique for extracting rain strokes to rain splatters.

While our basic implementation of rain is realistic and allows real-time rendering, extensions can be incorporated. For example, the influence of wind can be added by applying the dynamic wind model to the raindrop and then deforming the rain matte accordingly. In addition, we did not consider other rain-related effects such as ripples on surfaces and flow of water. Other approaches have been proposed for these (e.g., [16]), and can be used in conjunction with our rendering technique.

In our current raindrop model, different refraction indices for different frequencies of light can be considered, e.g., pre-computing the transfer vectors for R, G, and B channels separately. This will not increase the computation load and the extra memory cost is just a few bytes. However, we found that the difference between using the R-G-B transfer vectors and just a single transfer vector is imperceptible.

Since PRT uses only the low frequency parts of both the rain transfer function and the environment map, complementary techniques are required to account for the missing high frequency components. In our implementation, we simply added back a portion of the background, as shown in (2). Doing the same for the environment map remains a topic for future investigation.

Since we assume we can precompute the environment map ahead of time, our technique is currently less applicable for games, where the dynamic scene can arbitrarily change with user interaction. The speed of computing the SH coefficients of the environment map would have to be substantially increased to enable interactive gaming with our rain synthesis technique. This is another direction we are investigating as well.

8 Conclusions

We have presented a new technique to render realistic-looking rain in real-time using a commodity graphics accelerator. Our technique is a hybrid approach that capitalizes on real footage of rain to extract the realistic appearance, while our particle system allows real-time rendering. By ap-

plying PRT to our particle system, we increase the realism to the rendered rain scene by taking into account environment lighting. The closed-form solution we derived for the raindrop transfer function makes its evaluation highly efficient. Our PRT also results in low memory costs. All these benefits make our technique for rendering rain a prime candidate for real applications such as movie post-production.

Appendix A: Transmittance and Reflectance of Light

For unpolarized incident light, the reflectance, i.e., the ratio of the reflected power to the incident power, is [7]

$$R(\theta_i) = \frac{(1-\rho^2)^2 m^2 + (1-m^2)^2 \rho^2}{[m(1+\rho^2) + \rho(1+m^2)]^2},$$

where $\rho = \eta_t/\eta_i$, $m = \cos\theta_t / \cos\theta_i$, η_i and η_t are the refractive indices of the medium of incidence and transmission, respectively, and θ_i and θ_t are the angles of incidence and refraction, respectively. Note that the angle θ_t of refraction is determined by Snell's law: $\eta_i \sin\theta_i = \eta_t \sin\theta_t$. The transmittance, i.e., the ratio of the transmitted power to the incident power, is $T(\theta_i) = 1 - R(\theta_i)$.

Appendix B: Outgoing Light Intensity after Multiple Reflection

Because the radii of raindrops follow the Marshall-Palmer distribution [9], the radii of a vast majority of raindrops are less than 1mm. As the shape of raindrops are very close to be spherical when their radii are less than 1mm [4], assuming the shape of raindrops to be that of a sphere is very reasonable. This assumption leads to an analytically computable outgoing light intensity even though light is refracted inside the raindrop many times.

Let I_{in} and $\mathrm{d}F_{\text{in}}$ be the intensity of the incident light and the incident flux, respectively. Without loss of generality, we choose the direction of incident light to be $(0, 0)$, i.e., it is parallel to the z-axis of the local spherical coordinate of the raindrop. Let L_n, I_n, and $\mathrm{d}F_n$ be the direction, intensity, and infinitesimal flux of outgoing light after the incident light is reflected inside the raindrop $(n-1)$ times, respectively.

Note that the light path must all lie in the plane determined by the center of the raindrop and the incident light (Figure 2(a)). The incident light in direction $(0, 0)$ can hit different positions of the raindrop, resulting in different incident angle γ and the azimuth angle ϕ of the plane. Therefore, the directions of L_n can be parameterized as (θ_n, ϕ) (Figure 2(a)), where θ_n is derived in the following process.

To compute I_1, by the definition of reflectance, we have:

$$\mathrm{d}F_1 = R(\gamma)\mathrm{d}F_{\text{in}}, \tag{3}$$

with $R(\gamma)$ being the reflectance at angle γ. Note that

$$\mathrm{d}F_{\text{in}} = I_{\text{in}} \cos\gamma \cdot \mathrm{d}\omega_{\text{in}} = I_{\text{in}} \cos\gamma \cdot \sin\gamma \mathrm{d}\gamma\mathrm{d}\phi, \tag{4}$$

$$\mathrm{d}F_1 = I_1 \mathrm{d}\omega_1 = I_1 \sin\theta_1 \mathrm{d}\theta_1 \mathrm{d}\phi, \tag{5}$$

where I_{in} is the intensity of the incident light, and ω_{in} and ω_1 are the infinitesimal solid angles of incident and outgoing light, respectively. As the incident light is in the direction $(0, 0)$, we have $\theta_1 = \pi - 2\gamma$. By combining (3), (4), and (5), we have

$$I_1 = \frac{1}{4} R(\gamma) I_{\text{in}}. \tag{6}$$

To compute I_n $(n > 1)$, we consider the light being reflected $(n-2)$ times inside the raindrop and refracted 2 times when leaving the raindrop. Therefore,

$$\mathrm{d}F_n = [T(\gamma)]^2 [R(\gamma)]^{n-2} \mathrm{d}F_{\text{in}}, \tag{7}$$

where

$$\mathrm{d}F_n = I_n \mathrm{d}\omega_n = I_n \sin\theta_n \mathrm{d}\theta_n \mathrm{d}\phi. \tag{8}$$

Using the assumption that the raindrop is spherical, we get

$$\theta_n = \Theta(\zeta_n), \tag{9}$$

where $\zeta_n = \pi - 2\gamma - (n-1)(\pi - 2\beta)$, $\Theta(x) = |\text{mod}(x + \pi, 2\pi) - \pi|$ restricts the angle x to be between 0 and π, and $\beta = \arcsin(\sin\gamma/\rho)$ is the refraction angle when the incident light refracts inside the raindrop for the first time (Figure 2(a)). Combining (4), (7), and (8), we have:

$$I_n = I_{\text{in}} [T(\gamma)]^2 [R(\gamma)]^{n-2} \frac{\sin\gamma\cos\gamma}{\sin\theta_n \dfrac{\mathrm{d}\theta_n}{\mathrm{d}\gamma}}.$$

With (9), we have

$$\begin{aligned} \sin\theta_n \frac{\mathrm{d}\theta_n}{\mathrm{d}\gamma} &= \left| \sin\zeta_n \frac{\mathrm{d}\zeta_n}{\mathrm{d}\gamma} \right| \\ &= \left| \sin[2(n-1)\beta - 2\gamma] \left[2(n-1)\frac{\mathrm{d}\beta}{\mathrm{d}\gamma} - 2 \right] \right|. \end{aligned}$$

Finally, we arrive at $I_n = \xi_n I_{\text{in}}$, where

$$\xi_n = \frac{[T(\gamma)]^2 [R(\gamma)]^{n-2} \sin(2\gamma)}{4 \left| \sin[2(n-1)\beta - 2\gamma] \left[\dfrac{(n-1)\cos\gamma}{\rho\cos\beta} - 1 \right] \right|}.$$

Appendix C: Outgoing Light Intensity Distribution Function

In the previous section, we derived the scattering directions (θ_n, ϕ) and the corresponding attenuation coefficients ξ_n, given the incident angle γ and azimuth angle ϕ of the incident light. Note that the incident light in direction $(0, 0)$ can hit any point on the raindrop (Figure 2(b)). So, the intensity of light reflecting toward direction (θ, ϕ), in the local spherical coordinate of the raindrop with the viewing direction being along the z-axis, is

$$I_s(\theta, \phi) = \sum_{n=1}^{\infty} \sum_{\theta_n(\gamma)=\theta} I_n(\gamma).$$

Thus, the outgoing light intensity distribution function is

$$D(\theta, \phi) = \frac{1}{\pi I_{\text{in}}} I_s(\theta, \phi) = \frac{1}{\pi} \sum_{n=1}^{\infty} \sum_{\theta_n(\gamma)=\theta} \xi_n(\gamma), \tag{10}$$

where $\xi_1(\gamma) = R(\gamma)/4$ due to (6). The denominator πI_{in} is required for normalization, since it is the value of the integral of I_s over all directions. To make the computation more efficient, we precompute an inverse table of γ's that satisfies $\theta_n(\gamma) = \theta$. ξ_n usually drops rapidly with increasing n. Empirically, we have observed that the first nine ξ_n's hold about 99.8% of the full power; as a result, we use only $\xi_1, \cdots, \xi_9$ in our implementation.

References

[1] ATI Demo. Artist-directable real real-time rain rendering in time rain rendering in city environments. http://www.ati.com/developer/techpapers.html#gdc06, 2006.

[2] ATI Inc. ToyShop. http://www.ati.com/developer/demos/rx1800.html, 2005.

[3] Z.-X. Feng, M. Tang, J.-X. Dong, and S.-C. Chou. Real-time rendering of raining animation based on the graphics hardware acceleration. In *Proceedings of the 9th Int'l Conf. on Computer Supported Cooperative Work in Design*, volume 2, pages 734–739, 2005.

[4] K. Garg and S.K. Nayar. Detection and removal of rain from videos. In *Proceedings of CVPR 2004*, volume 1, pages 528–535, 2004.

[5] K. Garg and S.K. Nayar. When does a camera see rain? In *Proceedings of ICCV 2005*, volume 2, pages 1067–1074, 2005.

[6] R. Green. Spherical harmonic lighting: The gritty details. http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.html, 2003.

[7] E. Hecht and A. Zajac. *Optics*. Addison-Wesley, Reading, Massachusetts, 1974.

[8] Y. Horry, K. Anjyo, and K. Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proc. of SIGGRAPH 1997*, pages 225–232, 1997.

[9] J.S. Marshall and W.M.K. Palmer. The distribution of raindrops with sizes. *J. of Meteorology*, 5:165–166, 1948.

[10] S.G. Narasimhan and S.K. Nayar. Vision and the atmosphere. *Int'l J. of Computer Vision*, 48(3):233–254, 2002.

[11] S.G. Narasimhan and S.K. Nayar. Shedding light on the weather. In *Proceedings of CVPR 2003*, volume 1, pages 665–672, 2003.

[12] F. Pitié, A.C. Kokaram, and R. Dahyot. N-dimensional probability density function transfer and its application to colour transfer. In *Proceedings of ICCV 2005*, volume 2, pages 1434–1439, 2005.

[13] P.-P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proc. of SIGGRAPH 2002*, pages 527–536, 2002.

[14] S. Starik and M. Werman. Simulation of rain in video. In *Proceedings of the 3rd Int'l Workshop on Texture Analysis and Synthesis*, pages 95–100, 2002.

[15] H. C. van de Hulst. *Light Scattering by Small Particles*. New York, Dover, 1981.

[16] H. Wang, P.J. Mucha, and G. Turk. Water drops on surfaces. *ACM Trans. on Graphics (SIGGRAPH)*, 24:921–929, 2005.

[17] T. Yoshihiko, K. Kensuke, and T. Katsumi. A method for rendering realistic rain-fall animation with motion of view. In *IPSJ SIGNotes Computer Graphics and CAD Abstract*, pages 105–005, 2001.

[18] D. Zdrojewska. Real time rendering of heterogenous fog on the graphics hardware acceleration. In *Proc. 8th Central European Seminar on Computer Graphics*, pages 95–101, 2004.

(a) Yard

(b) Gateway

(c) Roof

(d) Roof with weak rain

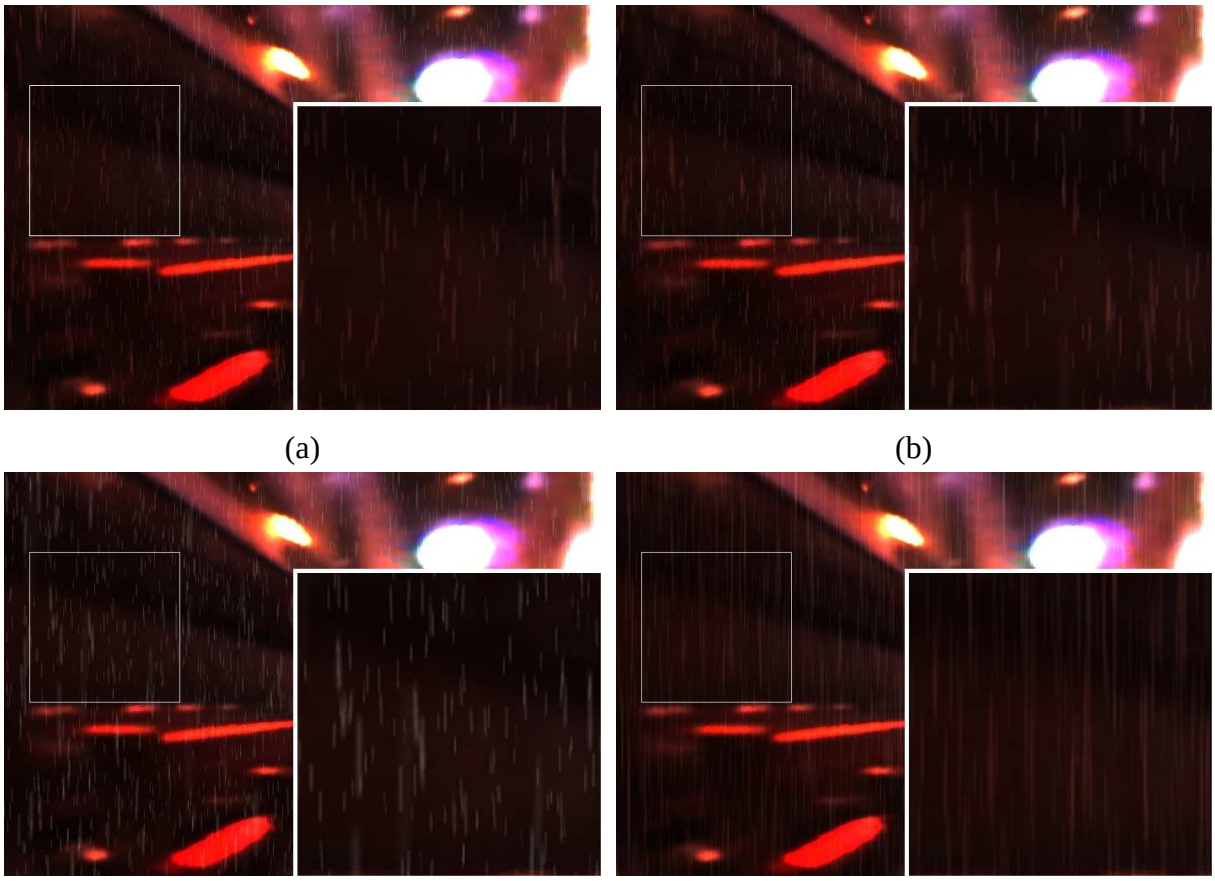
(e) Roof with medium rain

(f) Roof with heavy rain

Figure 5: *More examples of rain synthesis. (a)-(b) Adding the rain to daylight scenes Yard and Gateway. Note that the rain appears rather different from those in Figures 4(b), although the same rain samples were used. (c) Original rainy Roof (rain is imperceptible). (d)-(f) User-controlled rain getting progressively heavier. At the bottom-left corner of each image of Roof is a close-up of the boxed area on the right.*

(a) (b)

Figure 6: *Synthetic scene example with rain.*



(a) (b)

(c) (d)

Figure 7: *Comparison of results with different techniques. (a) Our technique with different stroke shapes, (b) ray tracing, (c) traditional particle system without considering environment lighting, (d) our technique with hand-drawn simple stroke shape. At the bottom-right corner of each image is a close-up of the boxed area on the left.*