Realistic real-time rain rendering

Pierre Rousseau, Vincent Jolivet and Djamchid Ghazanfarpour

Laboratoire MSI, Université de Limoges, France

Abstract

Real-time rendering of virtual weather conditions has been investigated in many papers. Inserting fog or snow in a scene is rather straightforward. Rain is one of the most encountered natural phenomena, but its rendering often lacks realism.

In this paper, we propose a realistic real-time rain rendering method using programmable graphics hardware. In order to simulate the refraction of the scene inside a raindrop, the scene is captured to a texture which is distorted according to optical properties of raindrops. This texture is mapped onto each raindrop. Our method also takes into account retinal persistence.

Key words: natural phenomena, rain, real-time rendering, graphics hardware, physical model.

1 Introduction

Until a few years ago, speed had usually a higher priority than realism for real-time applications. Nowadays, with the tremendous possibilities of current graphics hardware, these two points become less and less antagonist. Real-time applications begin to have new goals: photo-realism, following physics laws, handling a large number of natural phenomena.

To achieve a high degree of realism in order to immerse the user in a visually convincing environment, developers introduce weather conditions in their applications. Fog rendering reduces the observable depth in the scene, speeding up the rendering process. It has already been introduced in computer graphics, even by a full hardware acceleration. Falling snow can be approximated as an opaque and diffuse material. Consequently, it can be realistically represented

Email address: rousseau@msi.unilim.fr (Pierre Rousseau).

using simple particle systems. But falling rain still lacks realism, although it is one of the most encountered weather conditions in real scenes.

Rain rendering methods can be divided into two main categories. Most video-games use particle systems and static textures, leading to a lack of realism. Physically-based methods ([SDY02], [KKY93], [KIY99]) intend to simulate low-motion raindrops on a surface. They generate accurate results, at a high computation cost. The technique we present here has the advantages of both kinds of methods, without their drawbacks.

This paper introduces a method for a realistic rain rendering at a high frame-rate, making use of programmable graphics hardware. In addition, this method is based on physical properties (geometrical, dynamic and optical) of raindrops. An image of the background scene is captured to a texture. This texture is mapped onto the raindrops according to optical laws by a fragment shader. We extend this method to take into account retinal persistence: quasi spherical raindrops appear like streaks. With this motion blur extension, we generate more visually realistic rain rendering.

After presenting the previous related works, we present the physical (geometrical, dynamic and optical) properties of raindrops. Then, we describe our method to render realistic raindrops in real-time, and explain how we handle retinal persistence. We also propose an extension to handle illumination of raindrops from light sources. Finally, we present our results before conclusion and future works.

2 Previous works

Real-time rendering:

In most video-games (for example *Unreal Tournament 2004*, *Need For Speed Underground 2*, ...), rain is rendered as a simple particle system, where each particle is a translucent white streak. This method is not very realistic, but allows users to have the impression of a rainy environment.

An interesting work for video-games has been developed by N. Wang and B. Wade for *Microsoft Flight Simulator 2004* [WW04]. A textured double cone is positioned around the observer. Textures of light rain, heavy rain, snow, etc. are scrolled on the double cone to give a motion impression. The cone is tilted to cope with the speed of the observer, to give him the impression that the drops fall towards him. This method is faster than particle systems, but does not allow any kind of interaction between rain and the environment. In addition, a texture has to be defined for every desired type of precipitation.

Physically-based methods:

Many studies ([SDY02], [KKY93], [KIY99]) have proposed methods to render a limited number of low-motion water-drops on a surface such as a windshield. These methods produce satisfying results but imply a high computation cost, partly because of an expensive simulation process.

Computer vision methods:

In the field of computer vision, [SW03], [GN03] and [GN04] have described techniques to add or remove rain from video. To validate this approach, [GN03] needs a precise theoretical model to understand the influence of rain in videos, and for this purpose, it describes a ray tracing method that generates highly accurate raindrops, but at a prohibitive cost.

Other methods:

Some other papers cannot be related to one of the three above categories. Langer *et al.* [LZK+04] have presented an image-based spectral synthesis method to render snow and rain, where the spectrum of a falling snow or rain texture is defined by a dispersion relation in the image plane, derived from linear perspective. This method does not run in real-time. Another work proposed by Yang *et al.* [YZZ04] presented a simple method for distorting an image of the background scene in order to give the impression of drops on a windshield by using a very low cost algorithm (based on visual observations) without any relation with physical properties of raindrops. A real raindrop flips the scene behind it, which this method does not do, hence a lack of realism.

3 Physical properties of raindrops

3.1 Shape, size and dynamics

The widely spread idea according to which raindrops are tear-shaped, or streak-shaped, is inaccurate. This impression is caused, as we will see in section 5, by the phenomenon of retinal persistence. Many papers (referenced in [Ros00]), prove that falling raindrops look more like ellipsoids. Small raindrops are almost spherical, and bigger raindrops get flattened at the bottom.

This shape is the result of an equilibrium between antagonist forces. Surface tension tries to minimize the contact surface between air and raindrop, which results in a spherical shape. Aerodynamic pressure tries to stretch the drop horizontally, and gives it an ellipsoidal shape.

Green [Gre75] has proposed a simple model, balancing surface tension with the effects of gravity, resulting in ellipsoid raindrop shapes. Beard and Chuang ([BC87], [CB90]) have presented a more complex and accurate model, based on a sum of weighted cosines, to distort a regular sphere, using the following equation:

$$r(\theta) = a\left(1 + \sum_{n=0}^{10} C_n cos(n\theta)\right) \tag{1}$$

where a is the radius of the undistorted sphere, located at the center of mass of the drop. The angle θ denotes the polar elevation, with $\theta = 0$ pointing vertically downwards. A few sample shape coefficients C_n are given in table 1.

	Shape co-efficients ($c_n \cdot 10^4$) for n =										
a (mm)	0	1	2	3	4	5	6	7	8	9	10
0.5	-28	-30	-83	-22	-3	2	1	0	0	0	0
1.0	-134	-118	-385	-100	-5	17	6	-1	-3	-1	1
3.0	-843	-472	-2040	-240	299	168	-21	-73	-20	25	24
4.5	-1328	-403	-2889	-106	662	153	-146	-111	18	81	31

Table 1
Shape coefficients C_n for cosine distortion (equation 1) [CB90].

Figure 1 shows typical raindrop shapes for common undistorted radii, computed from equation 1.

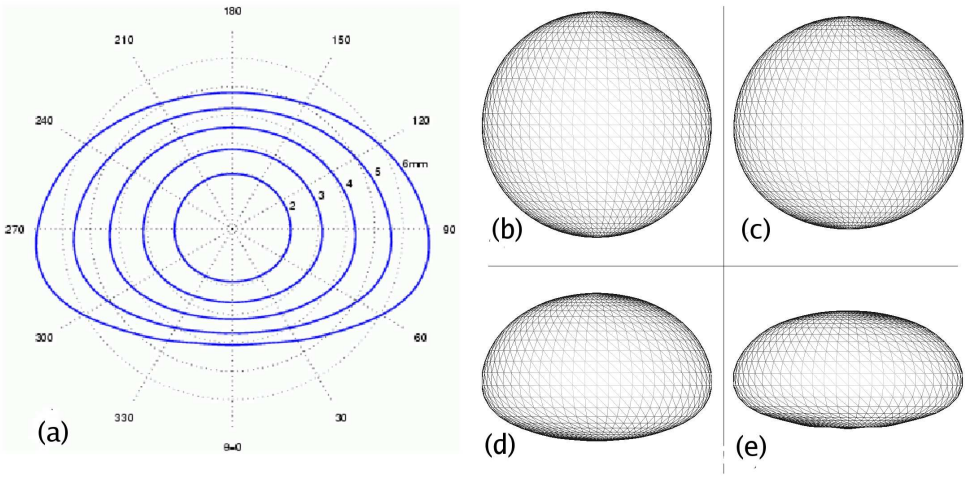

Fig. 1. Shape of drops. (a) Compared shapes of raindrops of radii R = 1mm, 1.5mm, 2mm, 2.5mm and 3mm [Ros00]. (b) Shape of a droplet of undistorted radius 0.5. (c) Shape of a droplet of undistorted radius 1.0. (d) Shape of a droplet of undistorted radius 3.0. (e) Shape of a droplet of undistorted radius 4.5.

The falling speed of a raindrop depends on its radius. Values presented in table 2 are speeds of raindrops which have reached their terminal velocities, when gravity and friction forces compensate. This velocity is quickly reached, and is the speed at which the drops are seen at ground level.

Spherical drops		Ellipsoidal drops			
radius (mm)	speed (m/s)	radius (mm)	speed (m/s)	radius (mm)	speed (m/s)
0.1	0.72	0.5	4.0	2.5	9.2
0.15	1.17	0.75	5.43	2.75	9.23
0.2	1.62	1.0	6.59	3.0	9.23
0.25	2.06	1.25	7.46	3.25	9.23
0.3	2.47	1.5	8.1	3.5	9.23
0.35	2.87	1.75	8.58	3.75	9.23
0.4	3.27	2.0	8.91	4.0	9.23
0.45	3.67	2.25	9.11		

Table 2
Speed of raindrops depending on their radii [Ros00].

3.2 Optical properties

In this paper, we do not intend to render rainbows (diffraction of light), so we do not need to take into account the wave character of light. It is physically correct to neglect the wave properties of light for drops much larger than the wavelength of light, which is the case here. We can instead focus on the properties defined by geometrical optics. In this approximation, light is considered as a set of monochromatic rays, which refract and reflect at interfaces between different propagation media.

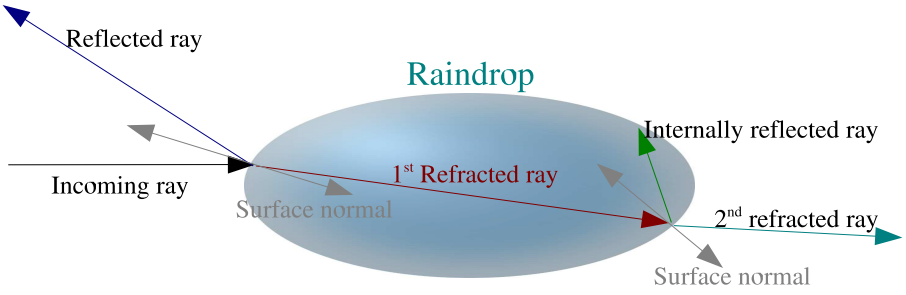

Fig. 2. Reflection / refraction of a ray in a raindrop.

At an interface, the law of reflection describes the directions of the reflected ray, and Snell's law describes the direction of the refracted ray. Directions of reflected/refracted rays are illustrated in Figure 2.

For a specific ray, given its angle of incidence onto the interface and its polarization, the ratio between reflection and refraction is given by the Fresnel factor.

In Figure 3, an example of refraction can be observed on a photograph (taken with a 1/1000 s shutter speed). The white dots on the photographed waterdrop are due to the camera flash.

Fig. 3. A photograph of a real drop refracting the background scene.

4 Real-time raindrop rendering

4.1 Hypotheses

The Fresnel factor computation demonstrates that reflection has only a significant participation to the color of a surface at grazing angles. For a raindrop, this means that reflection is only visible on the border of the drop. In our application, since a raindrop appears rather small on the screen, and reflection is visible only in a small part of each raindrop, it is reasonable to neglect the reflection participation to the appearance of the raindrop, and focus on correct refraction.

Raindrops are rendered as billboards [MS95] (small quads always facing the camera); the outline shape of the raindrops is given by a mask pre-computed from equation 1, for the desired raindrop radius. The computation of the mask is explained further in section 4.2. The appearance of each raindrop is computed inside a fragment shader.

4.2 Description of the method

The image perceived through a water-drop is a rotated and distorted wide angle image of the background scene, as illustrated in Figure 3. To simulate this effect, we use the render-to-texture facilities of graphics hardware to obtain a texture which will be map onto each raindrop in a fragment shader.

In a pre-computation step, we generate a mask for the desired radius, and save it into a texture.

During runtime, the appearance of each pixel of a raindrop is computed with the following process:

- Capture the scene to a wide angle texture.
- Using the mask, determine if the pixel is inside or outside the raindrop.
- If it is inside, use the mask to determine the direction of the refracted vector.
- Find the position in the captured texture, when there is no change in the direction of the incoming ray.
- In image space, add the refracted vector to the position found in the previous point.
- Extract the desired pixel at this location.

Pre-computation of the mask

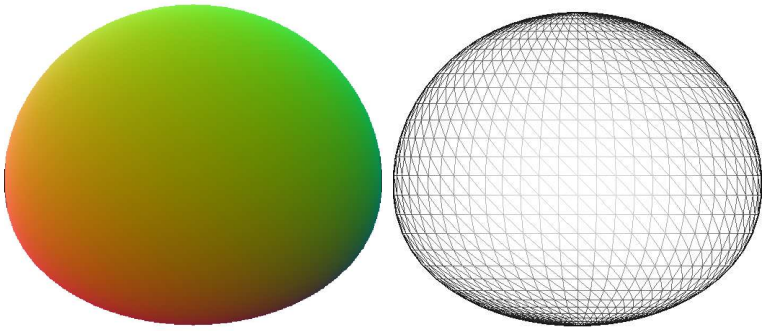

Fig. 4. Left: A mask texture pre-computed for a raindrop of radius 1.5mm. Right: A three-dimensional view of a raindrop of radius 1.5mm.

An auxiliary program uses equation 1 to compute the three-dimensional shape of a raindrop whose radius is given as a parameter. For each pixel of this shape, the refraction vector is pre-computed and saved into a texture (Figure 4). The mask can also be obtained from an arbitrary three-dimensional raindrop model. In the fragment shader, the mask is used at the same time to give the raindrop its shape and to determine the refraction vector at the low cost of a simple texture lookup. Instead of pre-computing this mask, it could have been possible to use Cg's *refract* function, and compute the refraction vectors at runtime. The drawback of this approach is that it limits the raindrops shapes to perfect spheres, as finding the point where the ray comes out of the raindrop implies a high computation cost for arbitrary shapes.

Capturing the scene to a texture

A camera is positioned at the same location as the observer, with the same orientation and with a very large angle of vision. This wide angle for the "Field Of View over y" (FOVy) parameter of the camera is explained in section 4.3. The texture generated by this camera is positioned on a plane behind the raindrops (as illustrated in Figure 5).

Determining the pixel color

For each pixel P_i of the raindrop, a fragment shader extracts the pixel which is refracted towards the observer from the captured texture (Figure 5). The fragment shader first determines which pixel P_o in the captured texture is the image of the scene object seen from the observer in the direction of P_i. Then the refraction vector is extracted from the mask texture, and combined to the location of P_o, to obtain pixel P_c, which gives the color of P_i.

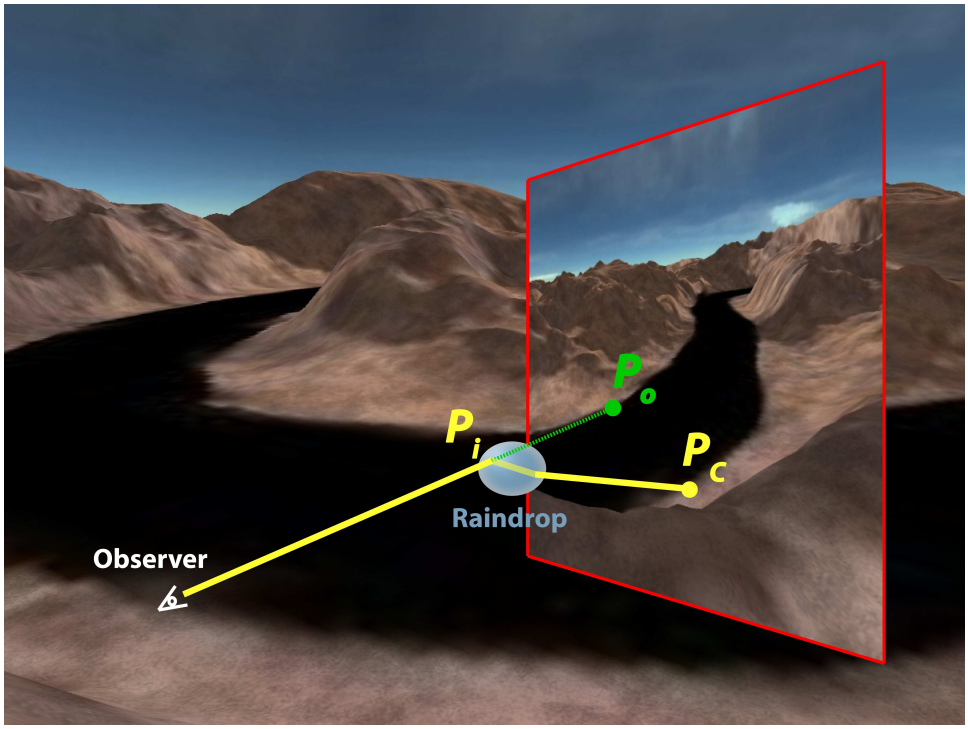

Fig. 5. Extraction of the raindrop pixel from the captured texture. The red quad delimits the plane onto which the generated texture is mapped. Rays coming from the observer to the raindrops are refracted towards the captured texture.

In Figure 6, we compare a water-drop simulated using our method (left) and an image of a real falling droplet (right). A photograph of the original scene was used as a background image for the simulated drop. The bottom images show a close view of the original and simulated drops. As the real drop just left the tap, its shape is not yet stabilized and is not perfectly spherical, and

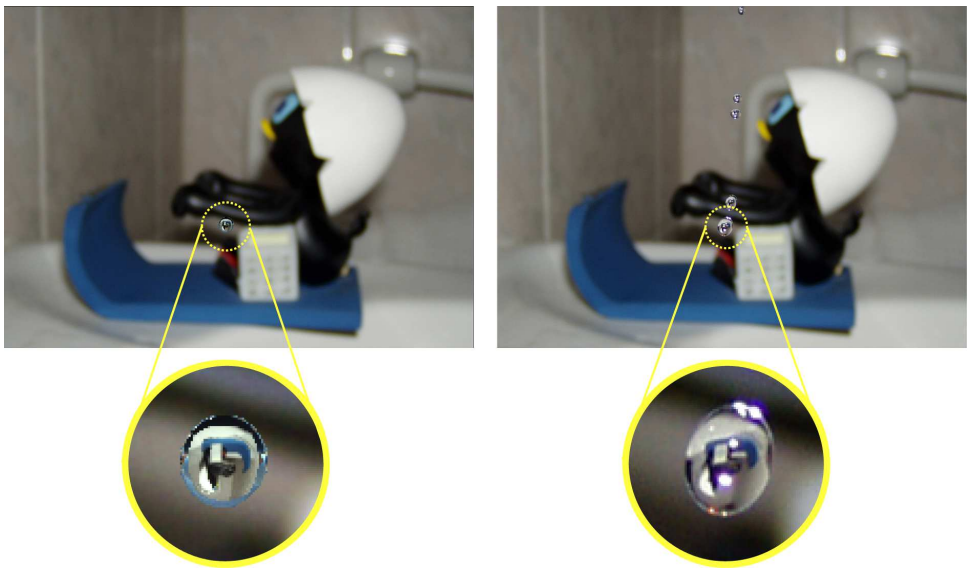

Fig. 6. Left: An image simulated with our method. Right: A photograph of a real raindrop.

so it does not behave exactly as the simulated one.

4.3 Physical parameters

FOVy of the camera

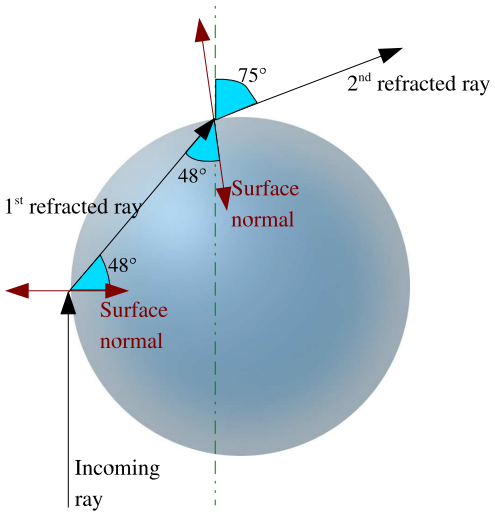

Fig. 7. Maximum refraction deviation for a raindrop.

The refraction index of air is 1, and that of water is 1.33. On the edges of a drop, where the refraction deviation is maximal, the angle between the ray coming from the observer and the normal to the surface is 90°, as illustrated in Figure 7. Using Snell's law, the angle between the incoming ray and the

internally refracted ray is 48°. The normal to the point where the ray comes out of the drop makes an angle of 6° with the original incoming ray. The refracted ray forms an angle of 48° with this normal, and so refracts back in the air with an angle of 81° to the normal (applying again Snell's law), and so 75° from the original incoming ray. The field of view of a raindrop is thus 150° wide. This value is sufficient in our application for the "Field Of View over Y" (FOVy) parameter of the camera capturing the scene to a texture.

Physical approximations

To obtain physically accurate results, we should perform a ray-tracing with all the objects in the scene, but this can hardly be done in real-time. The fact that we use only one texture for all the raindrops introduces a small approximation to physics laws, which implies a tremendous increase in the rendering speed. As it is not generated at the exact location of the drops, the texture does not contain what the drop really "sees". This can result in seldom cases in undetected occlusions, or in additional distortion in the texture mapping. In rain simulation, drops are very small and move very fast, and this approximation is not a major drawback.

5 Retinal persistence

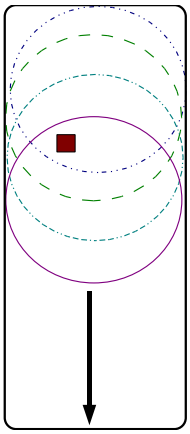

Fig. 8. The pixel indicated in red receives a contribution from all the successive positions of the raindrop.

We defined a general-purpose model for rain simulation, which doesn't take into account perception from an observer. Because of retinal persistence, a human eye or camera often perceives raindrops like streaks. Two slightly different phenomena can be observed: a camera captures its picture in discrete time, while human eye operates in continuous time.

In a photography or a motion picture, raindrops appear like streaks due to the shutter speed of the camera. While the shutter is opened, a drop falls down a few centimeters, and impresses the film on a short vertical distance. This effect is usually called "motion blur". It would not be visible for an ideal camera using an infinitesimal shutter speed.

The eye observing real rain behaves differently, for the same result. An eye does not have a shutter, but when an image forms on the retina, it takes 60 to 90 milliseconds to fade away. During this time lapse, the drop keeps falling, and all its different positions compose a continuous sequence of images on the retina, producing this persistence effect.

Human eye is not used to seeing almost spherical drops; our model, although it is physically correct, seems to lack realism. We extended our model to take into account retinal persistence, and generate streaks based upon our accurate raindrop model.

To simulate this effect, our rain particles are reshaped into vertical streaks. Each pixel of a streak receives the contribution of the successive positions of the drop, as illustrated in Figure 8. The fragment shader we use is modified in the following way: for each pixel;

- Compute the refracted pixel of a few sample positions of the drop
- Perform a mean of those values.
- Lower alpha value, since each streak is the result of one moving drop.

6 Results

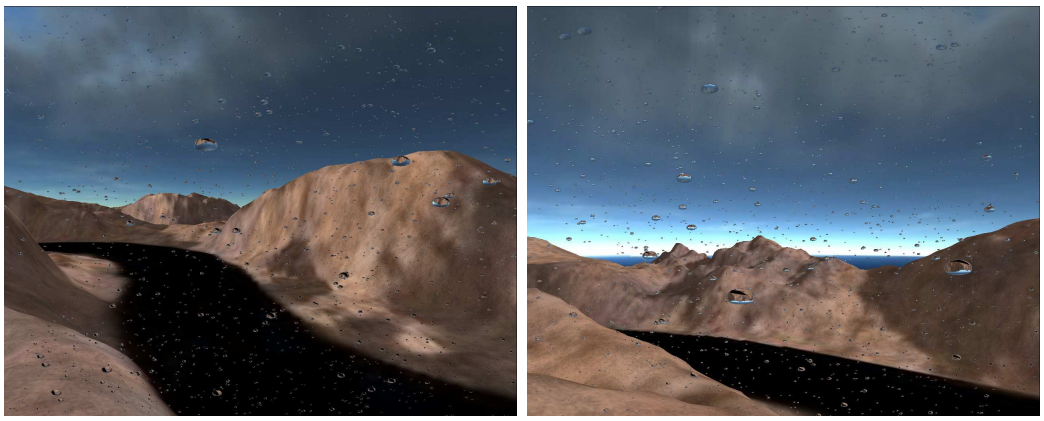

Fig. 9. Two images generated with our method.

The main bottleneck in our application is the handling of the particle system. On a PC with a 2600+ AMD CPU and an nVidia Geforce 6800 GT video card, our method generates more than 100 frames per second, when using

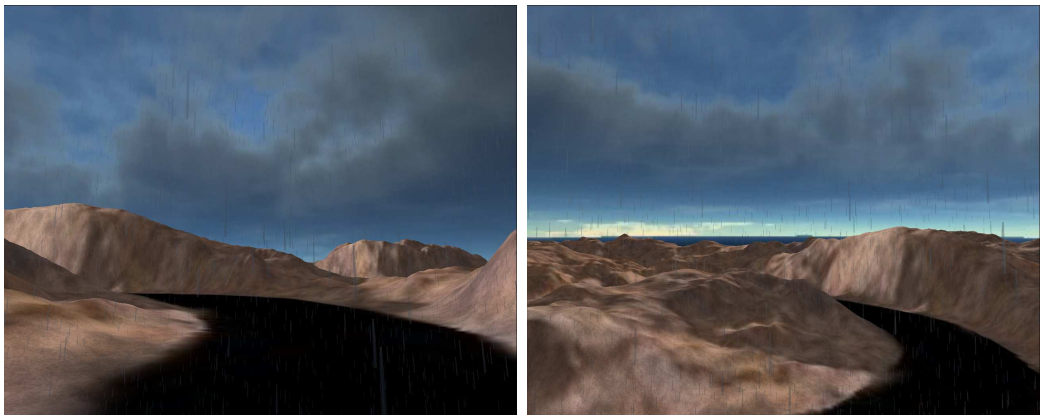

Fig. 10. Two images generated using our retinal persistence extension.

5000 particles. Increasing the particle count to 20 000, reduces this frame-rate to 25 images per second. This bottleneck should be removed using a hardware implementation of the particle system, as proposed by Kolb *et al.* [KLRS04].

5000 particles are sufficient to provide a realistic rain impression for large raindrops or streaks. When using very small raindrops (below a radius of 1mm), 10000 particles are required for a realistic rain impression.

Figure 9 shows a scene including 5000 raindrops of radius 4.5 mm, animated at a frame-rate of 100 Hz.

Figure 10 shows results obtained with our retinal persistence extension. 5000 raindrops of radius 1.5 mm are animated at 70 Hz. This extension implies a higher computing cost (depending on the number of samples used), but needs less particles to produce a realistic effect.

Since rain is an animated phenomenon, it is better observed on videos than on static images. Videos of our method in action can be downloaded from *http://msi.unilim.fr/∼rousseau/rain.html*.

7 Extension: light/raindrop interaction

When rain falls near street lights or head lights, real raindrops present reflects of the color of the light sources.

The optical laws presented in section 3.2 still apply when a light source is positioned in the scene. When the observer is close to a light source, the rays coming from this source have a far greater intensity than rays coming from anywhere else in the scene. Using the method described above, a light source positioned behind the observer would not have any influence on the

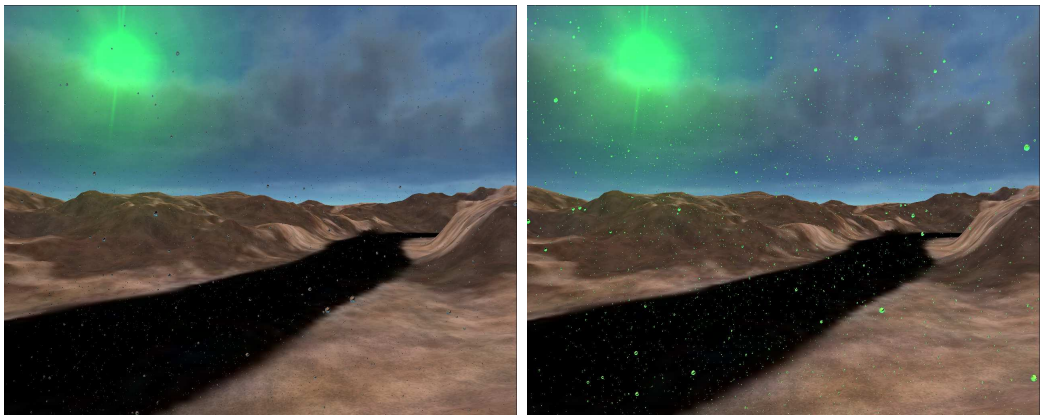

Fig. 11. A light source modifies the color of the raindrops. Left: Without the light/raindrop extension. Right: With the extension activated.

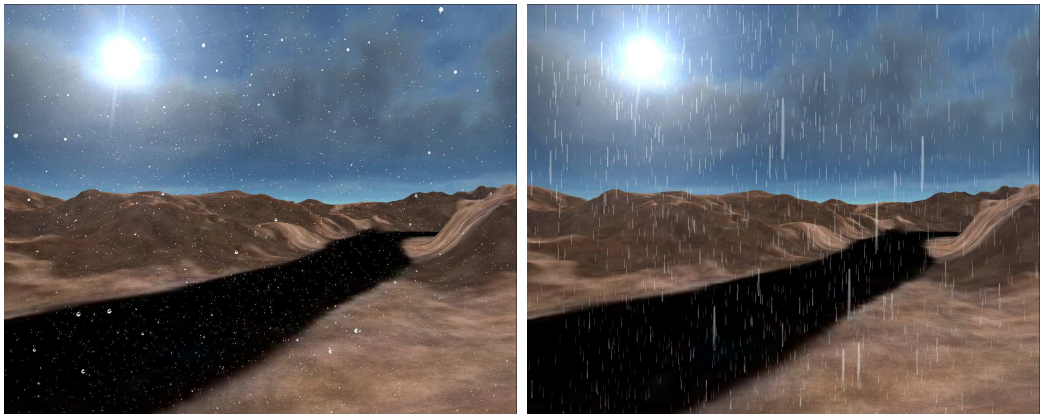

Fig. 12. Left: Light/raindrop extension, with a white colored light. Right: Using the retinal persistence extension.

generated raindrops, because our model does not handle reflection (which most of the time, is negligible, see section 4.1). In the case of a close light source, reflection and internal reflection or refraction cannot be ignored, since they have an important participation to the appearance of the drop (considering the intensity of rays coming from the light source.)

Computing all the internal reflections of light rays would be the best way to generate physically satisfying images, but it cannot be done in real-time. We simulate this effect by modifying the color of the raindrops pixels, based on the distance between the raindrop and the light source, using the following empiric formula:

$$C_f = (C_o * C_{amb}) + (C_o * C_{diff} * \frac{\overrightarrow{V_{lightToPixel}} \cdot \overrightarrow{V_{pixelNormal}}}{D_{lightToDrop}}) \tag{2}$$

Where:

- C_f is the final color of the pixel,
- C_o is the color extracted from the texture,
- C_{amb} is the ambient light color of the scene,
- C_{diff} is the diffuse color of the light source,
- $\overrightarrow{V_{lightToPixel}}$ is the direction of the ray going from the light source to the considered pixel,
- $\overrightarrow{V_{pixelNormal}}$ is the direction of the normal to the drop at the considered pixel,
- $D_{lightToDrop}$ is the distance between the light source and the considered pixel.

This formula gives visually satisfying results, as illustrated in Figure 11; Figure 12 shows images generated with a white light source, using our retinal persistence extension in the right image. In our implementation of this technique, we can handle two point light sources at the same time, without any significant performance loss.

8 Conclusion and future works

We have developed a physically based real-time model for rendering raindrops. We extended this model to handle retinal persistence and light sources. Our model produces better results than the usual particle systems using static textures which are often used in video-games. It achieves a much faster rendering speed than the existing physical based models.

We believe that our model can be widely used in video-games or driving simulators as it generates visually convincing results at a high frame-rate.

For perfectly accurate results, the two possible techniques are either a complete ray-tracing on each object of the scene, or a dynamic generation of a cubic environment map for every single raindrop. Both of these methods cannot run in real-time, at least with current graphics hardware. Our model introduces some approximations to these methods. Consequently, it is not physically completely accurate but allows a real-time high frame-rate execution, and is a good approximation of the images which would be obtained by other methods.

In future works, we will try to add reflection to our model in order to generate even more realistic raindrops viewed from a close distance. The equation we use to take into account light sources is subject to further improvements. Finally, we will also develop an alternate simpler model, to be used for farther raindrops, whose size on screen falls below a pixel.

Acknowledgments

This work is supported by GameTools project of European Community (contract number 004363).

References

[BC87] K. V. Beard and C. Chuang. A new model for the equilibrium shape of raindrops. *J. Atmos. Sci.*, 44(11):1509–1524, 1987.

[CB90] C. Chuang and K.V. Beard. A numerical model for the equilibrium shape of electrified raindrops. *J. Atmos. Sci.*, 47(11):1374–1389, 1990.

[GN03] Kshitiz Garg and Shree K. Nayar. Photometric model of a rain drop. Technical report, Columbia University, 2003.

[GN04] Kshitiz Garg and Shree K. Nayar. Detection and removal of rain from videos. In *CVPR (1)*, pages 528–535, 2004.

[Gre75] A. W. Green. An approximation for the shapes of large raindrops. *J. Appl. Meteor.*, 21:1578–1583, 1975.

[KIY99] Kazufumi Kaneda, Shinya Ikeda, and Hideo Yamashita. Animation of water droplets moving down a surface. *Journal of Visualization and Computer Animation*, 10(1):15–26, 1999.

[KKY93] K. Kaneda, T. Kagawa, and H. Yamashita. Animation of water droplets on a glass plate. In *Computer Animation*, pages 177–189, 1993.

[KLRS04] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 123–131, New York, NY, USA, 2004. ACM Press.

[LZK+04] Michael S. Langer, Linqiao Zhang, Allison W. Klein, Aditya Bhatia, Javeen Pereira, and Dipinder Rekhi. A spectral-particle hybrid method for rendering falling snow. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering)*. ACM Press, june 2004.

[MS95] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Symposium on Interactive 3D Graphics*, pages 95–102, 211, 1995.

[Ros00] Oliver N. Ross. Optical remote sensing of rainfall micro-structures. Master's thesis, Freie Universität Berlin, 2000. in partnership with University of Auckland.

[SDY02] Tomoya Sato, Yoshinori Dobashi, and Tsuyoshi Yamamoto. A method for real-time rendering of water droplets taking into account interactive depth of field effects. In *Entertainment Computing: Technologies and Applications, IFIP First International Workshop on Entertainment Computing (IWEC 2002)*, volume 240 of *IFIP Conference Proceedings*, pages 125–132. Kluwer, 2002.

[SW03] S. Starik and M. Werman. Simulation of rain in videos. In *3rd international workshop on texture analysis and synthesis (Texture03)*, pages 95–100, 2003.

[WW04] N. Wang and B. Wade. Rendering falling rain and snow. In *ACM SIGGRAPH 2004 Technical Sketches Program*, 2004.

[YZZ04] Yonggao Yang, Changqian Zhu, and Hua Zhang. Real-time simulation: Water droplets on glass windows. *Computing in Science and Eng.*, 6(4):69–73, 2004.