

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, and interaction with light sources.

Key words: natural phenomena, rain, physically based modeling, real-time rendering.

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 more and more compliant. Real-time applications begin to have new goals, which they can handle without sacrificing performance: photo-realism, following physics laws, handling a large number of natural phenomena. As long as the frame-rates requirements are satisfied, there is no limitation to the range of effects that can be added to real-time applications.

A high degree of realism is required to immerse the user in a visually convincing environment. For this purpose, 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

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

graphics, even by a full hardware acceleration ([Bir03]). Falling snow can be approximated as an opaque and diffuse material. Consequently, it can be realistically represented using simple particle systems ([LZK⁺04]). 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 computational 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. We also propose another extension to the described technique, to handle illumination of the raindrops by light sources.

After presenting the previous related work, we present the physical (geometrical, dynamic and optical) properties of raindrops. Then, we describe our method to render realistic raindrops in real-time and propose extensions to handle retinal persistence as well as illumination of raindrops by light sources. To animate our raindrops, we extend hardware particle simulation methods, specializing them to rain animation. Finally, we present our results before conclusions and future works.

2 Previous work

Rain rendering has been investigated in some papers, and becomes common in video games. Most existing techniques aim at high speed rendering, or physical accuracy. We intend to make these antagonist goals compliant, proposing a method relying on physical properties of raindrops for high quality results (close to those obtained by ray-tracing methods), and nevertheless not increasing the computational cost of those currently used in real-time rendering. Some papers in related research fields also draw our attention and are of interest to our purpose, though not strictly dealing with water or rain rendering.

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. The method proposed in this paper aims at providing more realism and flexibility than the methods presented above, without increasing their computational cost.

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, using dynamic cube-map generation. These methods produce satisfying results but imply a high computational cost, partly because of an expensive simulation process. Our technique reduces the cube-map rendering to only one capture, which we prove sufficient to compute refraction through raindrops.

More recently, [WMT05] proposed a method to animate water-drops on arbitrary surfaces, allowing drops to merge or split. This technique produces high quality results at a prohibitive cost.

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. The technique described in this paper is based on physical laws, thus providing a higher degree of realism.

2.2 Related research fields

Real-time refraction:

Refraction through translucent objects using impostors to avoid expensive ray-tracing computation has been investigated in [Wym05] and [SKALP05]. Slow motion refractive objects are rendered with dynamic environment-map generation, correcting the inherent drawback that objects appear the same wherever they are positioned. These methods produce high quality results for a low computational cost and can be extended to the case of refractive raindrops.

Light interaction:

Rendering of water in general implies refraction, reflection and interaction with light. Premoze and Ashikmin ([PA00]) render large amounts of water taking into account light transport above and under the surface. Thon [Tho01] considers caustics, absorption and diffusion inside the water. The methods described use ray-tracing, and consequently do not run in real-time, but indicate the level of realism to seek. Iglesias [Igl04] presented a historical survey of water modeling and rendering techniques, providing many useful references.

Some articles do not deal with rain or even water, but still present high interest for our purpose. For example, [SFD00] or [GS04] present techniques implying computation of external and internal reflection, refraction, and even light polarization (in [GS04]) in order to realistically render gemstones. We cannot use the same approach in the case of rain rendering, mainly because a raindrop can hardly be considered as a polygonal object (or it would need to be highly tessellated, which would result in bad frame-rates with these techniques).

Illumination in participating media:

Real-time computation of light transport through participating media has been widely investigated, for example considering clouds [Har03], haze, mist or fog (*e.g.* in [Bir03], [SRNN05]). They notably introduce single or even multiple scattering in real-time through participating media.

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:

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

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.

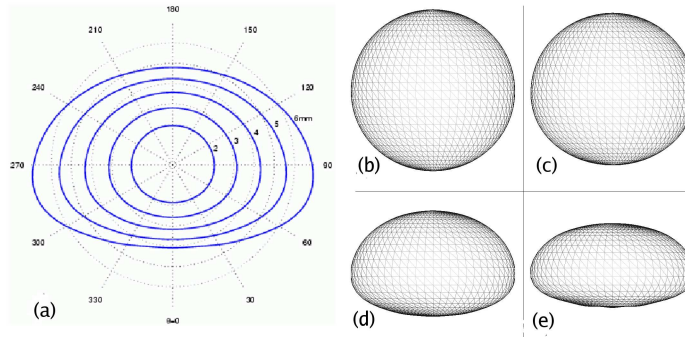


Fig. 1. Shape of drops. (a) Compared shapes of raindrops of radii $R = 1\text{mm}$, 1.5mm , 2mm , 2.5mm and 3mm [Ros00]. (b) Shape of a droplet of undistorted radius 0.5 . (c) Shape of a droplet of undistorted radius 1.0 . (d) Shape of a droplet of undistorted radius 3.0 . (e) Shape of a droplet of undistorted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

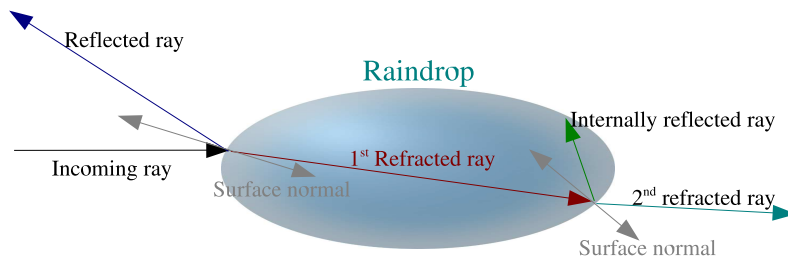


Fig. 2. Reflection / refraction of a ray in a raindrop.

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. More details about these phenomena can be read in [Gla94].



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

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 water-drop are due to the camera flash.

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 a raindrop 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 mapped 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, for each frame rendered, the scene is captured to a wide angle texture. The appearance of each pixel of a raindrop is computed with the following process:

- 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

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

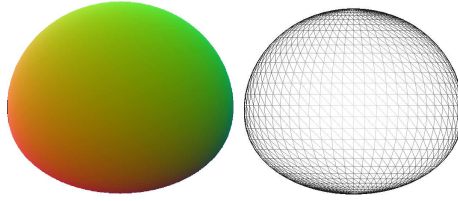


Fig. 4. Left: A mask texture pre-computed for a raindrop of radius 1.5mm. Right: A three-dimensional view of a raindrop of radius 1.5mm.

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 .

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 so it does not behave exactly as the simulated one.

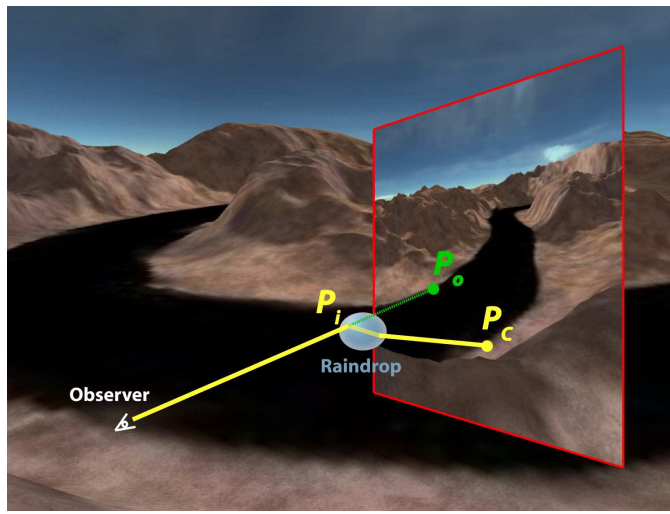


Fig. 5. Extraction of the raindrop pixel from the captured texture. The red quad delimits the plane onto which the generated texture is mapped. Rays coming from the observer to the raindrops are refracted towards the captured texture.

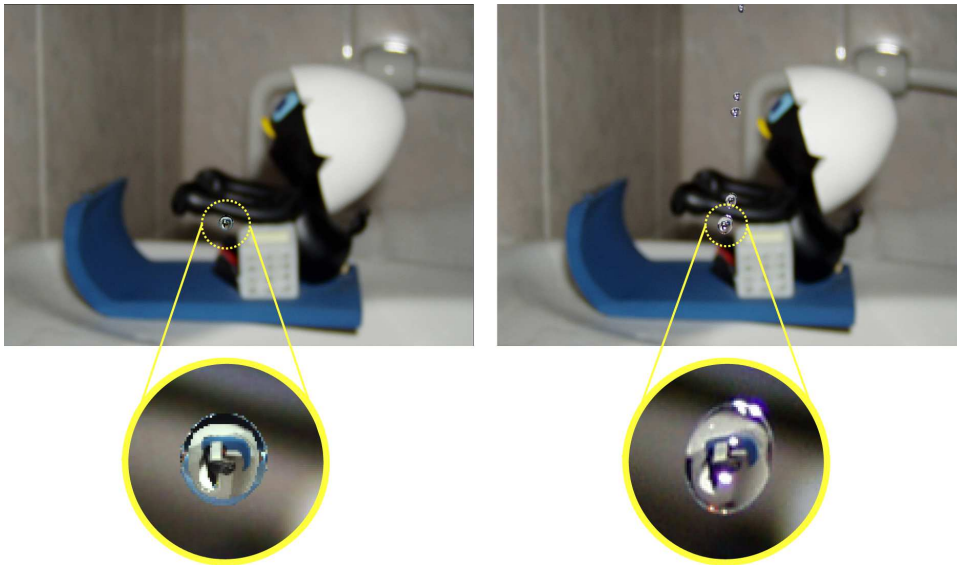


Fig. 6. Left: An image simulated with our method. Right: A photograph of a real raindrop.

4.3 Physical parameters

FOV_y of the camera

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

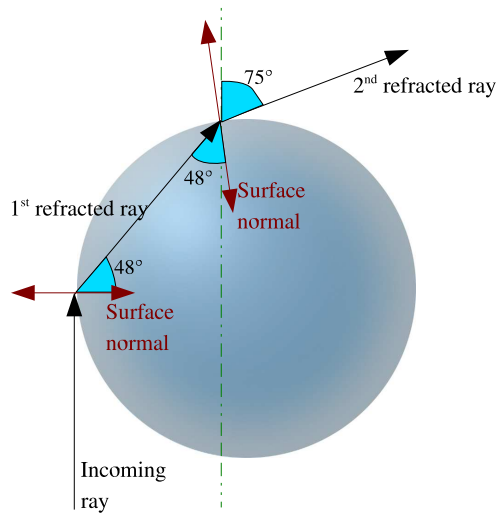


Fig. 7. Maximum refraction deviation for a raindrop.

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". In some cases, this can result 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 Extension: retinal persistence

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 the human eye operates in continuous time.

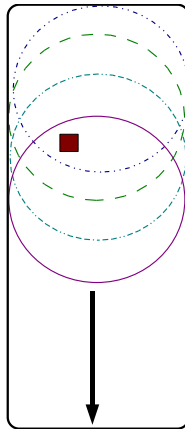


Fig. 8. The pixel indicated in red receives a contribution from all the successive positions of the raindrop.

In a photograph 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, with 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.

The 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 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 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, external and internal reflection 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.)

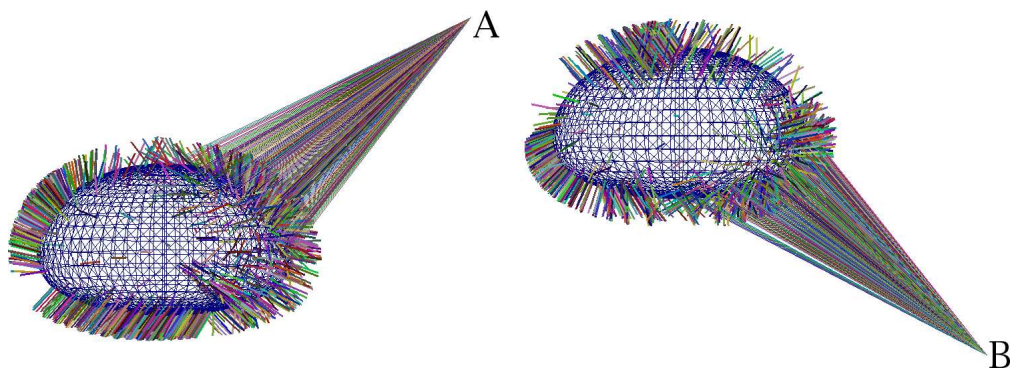


Fig. 9. Distribution of rays coming out from a raindrop (undistorted radius: 3.5mm). Rays sent from the A and B point light sources enter the drop from the right of the image, reflect internally up to three times, and refract out of the drop.

Figure 9 shows rays traced through a drop. Short colored sticks show the direction along which rays are refracted out of the drop, after 0, 1, 2 or 3 internal reflections (when most of the intensity of the original ray has been refracted back in the air.) We can observe that even after three internal reflections, the directions of the outgoing rays can be classified in three groups:

- back in the direction of the original ray (external reflection or third internal reflection)
- opposite side of the drop, upward
- opposite side of the drop, downward

It appears that very few rays come out of the drop sideways.

Computing a few internal reflections of light rays would be the best way to generate physically satisfying images, but it can hardly be done in real-time, and would not be compliant with our billboard model. Considering the above

observations, we propose to simulate light interaction with a raindrop by modifying the color of raindrops pixels, based on the distance between the raindrop and the light source, using the following empirical formula:

$$\begin{aligned}
A_1 &= C_L * DistFact * \overrightarrow{V_{P \rightarrow E}} \cdot \overrightarrow{V_N} \\
A_2 &= \max(0, \overrightarrow{V_{P \rightarrow L}} \cdot \overrightarrow{V_N}) \\
A_3 &= \max(0, (-\overrightarrow{V_{P \rightarrow L_{2D}}}) \cdot \overrightarrow{V_{N_{2D}}} * (1 - (-\overrightarrow{V_{P \rightarrow L}}) \cdot \overrightarrow{V_N})) \\
C_F &= (C_O * C_{amb}) + \sum_{lights} \left(A_1 * (A_2 + A_3) \right)
\end{aligned}$$

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_L is the color of the light source,
- $\overrightarrow{V_{P \rightarrow E}}$ is the direction of the ray going from the pixel to the eye,
- $\overrightarrow{V_N}$ is the direction of the normal to the drop at the considered pixel,
- $\overrightarrow{V_{P \rightarrow L}}$ is the direction of the ray going from the light source to the considered pixel,
- $\overrightarrow{V_{P \rightarrow L_{2D}}}$ is the normalized projection of the previous ray on the horizontal plane,
- $\overrightarrow{V_{N_{2D}}}$ is the normalized projection of the normal to the pixel on the horizontal plane,
- $DistFact$ is an attenuation factor based on the distance between the light source and the considered pixel.

A_1 defines the maximum color modification which can be applied to the pixel. We can notice in figure 9 that most rays leave the drop in a direction close to the normal at the outgoing point; this explains why A_1 gives more intensity to the pixels whose normal is facing the user.

A_2 is used for external reflection; in this case, the ray goes back in its original direction and the dot product is positive. This ensures that the more directly reflected rays will receive the more energy.

A_3 is used for internal reflections. The dot product in the horizontal plane ensures that the rays are not refracted sideways, which should not be the case according to Figure 9. The second dot product favors rays going perpendicularly upward or downward.

This formula gives visually satisfying results, as illustrated in the results shown in section 8. In our implementation of this technique, we handle two point

light sources at the same time, without any significant performance loss (this number could be extended further very easily.)

This extension is fully compliant with the retinal persistence extension described in section 5.

7 Hardware raindrop motion simulation

7.1 Description:

Animating falling rain is obviously a particle system matter. In our application, it appeared that at least 5000 particles are needed to produce a realistic impression of rain. Handling the animation via the CPU implies transferring all the particles to the graphics card for each frame. This proves a major bottleneck, reducing frame-rates drastically when the particle count increases.

To overcome this limit, we used the approach described in [KLRS04] and [KSW04]. These papers propose hardware implementations of particle systems. Particle positions are stored in floating point textures and updated through shader programs. This simply eliminates data transfers between CPU and GPU regarding particle simulation, all the required computation being made using the graphics hardware.

7.2 Implementation:

Our implementation of this feature is slightly different from those described in [KLRS04] and [KSW04]. We restricted our work to a specialized rain simulator and did not implement a general purpose particle system as described above. We made the assumptions that all particles share the same speed, are not subject to collisions, and do not need to be sorted (as the particles are not very dense and move very fast). Contrary to the other papers (which just mention it), we made use of the *Shader Model 3.0* functionality called *Vertex Texture Fetch*. This enables textures to be read from a vertex shader.

Particles are considered to evolve in a "rain box" surrounding the observer, and moving with him. This provides cheap clipping and culling facilities. Positioning the box wisely, most drops are always inside the view frustum of the observer, and at a reasonable distance where it still makes sense to render them.

We use two textures for this simulation. The first one indicates the position

of the particles. The second texture consists of only a few pixels, containing the information needed to update the position texture. The position texture is updated in the graphics hardware, while the information texture is generated procedurally in software for each frame, thus allowing the user to interact with the simulation's behavior.

The simulation process can be summarized as follows:

- At startup, store the needed particles in GPU memory, pretending that they cannot be culled and that they will never move.
- Update information texture.
- Update position texture.
- For each particle, modify their actual position in a vertex shader, according to the position texture.
- Render each particle, with the material described above.

7.3 Information texture:

This texture is procedurally defined for each frame, and sent to the GPU. This enables the user to modify particle parameters dynamically, for the negligible cost of transferring a few pixels to the GPU each frame. This one-dimensional texture contains:

- The time elapsed since last frame was rendered, used to insure a constant speed animation.
- The particle count, size of the position texture, and size of the width of the particles.
- The position of the viewer.
- A vector indicating the displacement of a drop per second. This enables the user to dynamically modify the speed and direction of the falling drops, for example introducing wind.
- The displacement of the user since last frame. Drops fall in a user centered reference frame, which means we need to compensate movements of the user so that drops fall direction does not depend on the user.
- Small random numbers, used when re-spawning particles to avoid a cyclic move.

7.4 Position texture:

In this texture, each pixel's red, green and blue values correspond to a particle's X, Y and Z coordinates.

When updating this texture, we simply add the displacement since last frame to the pixel's color components. If the resulting position is outside of the box, we reposition the particle close to the face of the box on the opposite side of the one where it came out. Typically, when a particle reaches the bottom, we simply throw it back to the top, slightly modifying its coordinates using the random factor read from the information texture.

7.5 Particle orientation:

Positioning a particle must take into account its orientation with regards to the observer. The initial position of a particle is simply discarded, and computing the position of each corner of a particle when updating the position texture could reveal time-consuming. We used a different approach, allowing for two different types of billboards to be rendered efficiently.

The crucial part of a vertex shader is the transformation of the object's position, from world space to screen space. In the case of standard raindrops, each drop should appear roughly spherical, and so should be facing the camera. The particle should then lie parallel to the image plane. We can very easily handle this, computing screen space position of a point particle, and then modifying the obtained screen coordinates. For instance, the top left corner of the particle will have its X coordinate lowered and its Y coordinate augmented. This will stretch the particle on screen.

In the case of streaks, a slight modification has to be applied; for instance, when the user looks upwards, the streaks should obviously appear vertical in world space and not in screen space. For this purpose, we simply modify the vertical components of each particle before the world-to-screen conversion, and the horizontal components after this conversion.

8 Results

In our experience, it appears that 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 at least are required for a realistic rain impression. 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 for a typical scene containing 5000 particles. The frame-rates mentioned below are achieved with the same configuration.

Figure 10 shows a typical rainy scene rendered by our application, with various

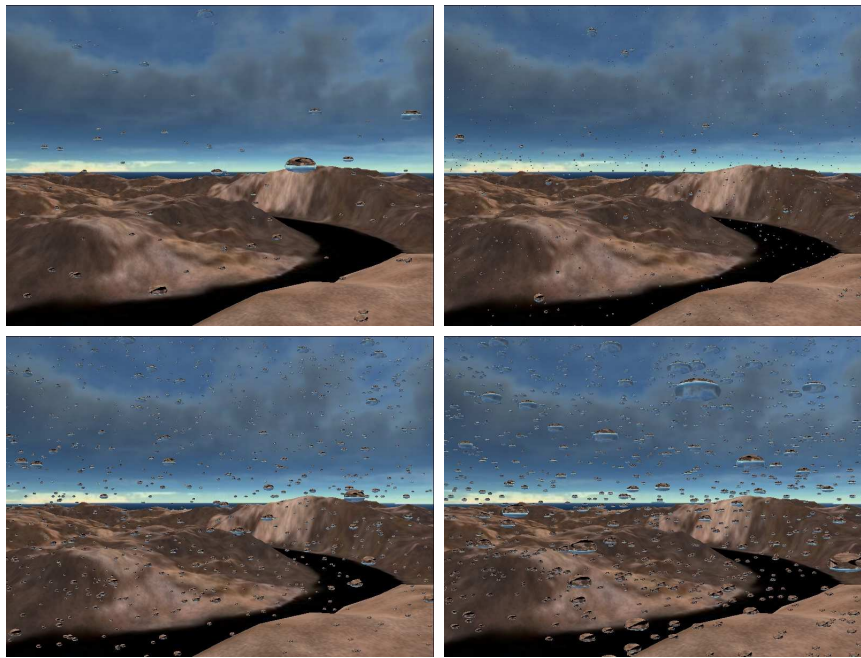


Fig. 10. Top left: 1000 large raindrops (undistorted radius: 4.5). Top right: 16000 small raindrops (undistorted radius: 1.0). Bottom left: 16000 medium raindrops (undistorted radius: 3.0). Bottom right: 16000 large raindrops (undistorted radius: 4.5).

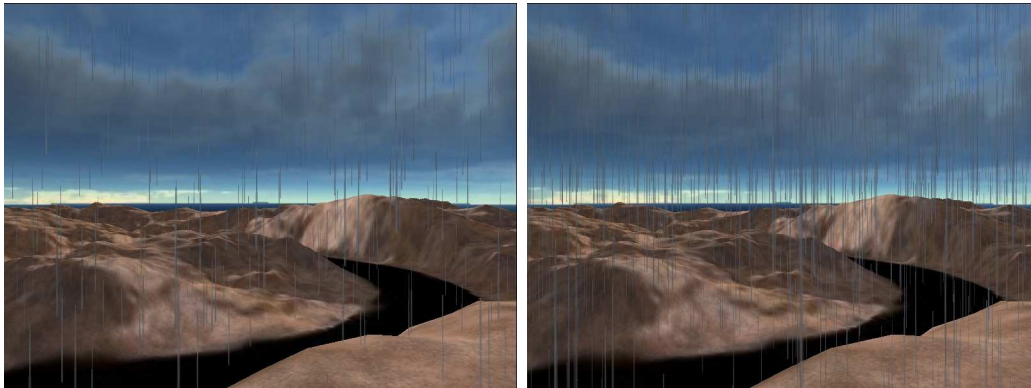


Fig. 11. Retinal persistence extension. Left: 4000 streaks (undistorted radius: 1.0). Right: 16000 streaks (undistorted radius: 0.5)

particle size and count. These images are rendered with frame-rates ranging from 35 Hz (in the worst case) to 170 Hz (with only one thousand particles).

Figure 11 shows results obtained with our retinal persistence extension. 4000 raindrops of radius 1.0 mm are animated at 60 Hz. This extension implies a higher computing cost (depending on the number of samples used), but needs fewer particles to produce a realistic effect.

Figure 12 illustrates the use of the light interaction extension. We can observe that the image on the right, which uses the extension, appears more visually

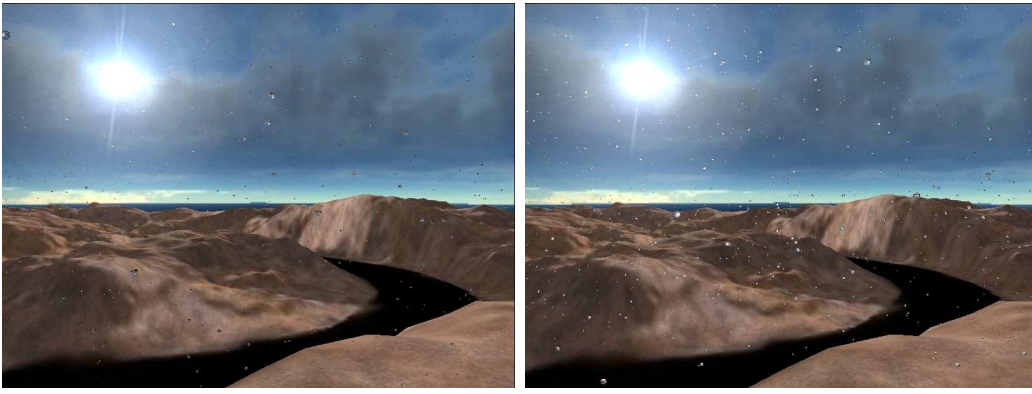


Fig. 12. Light interaction extension (10000 drops, undistorted radius: 1.0). Left: Without the light/raindrop extension. Right: With the extension activated.

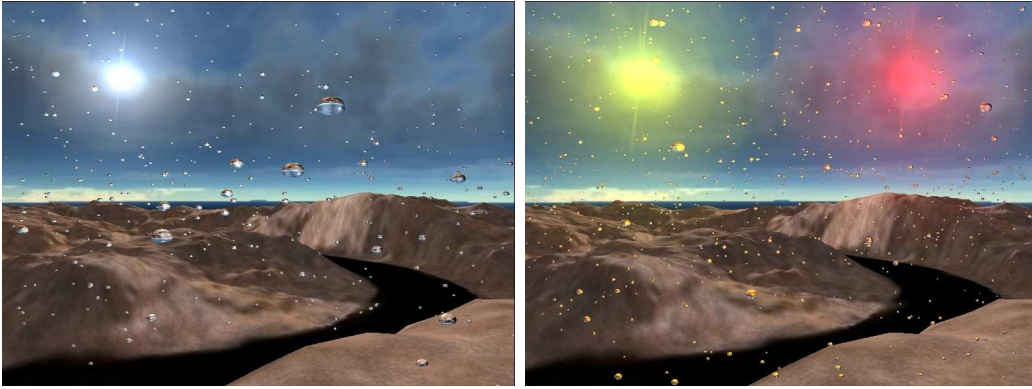


Fig. 13. Left: 4000 medium drops (undistorted radius : 2.5) illuminated by a white light source. Right: two colored light sources illuminating 10000 drops.

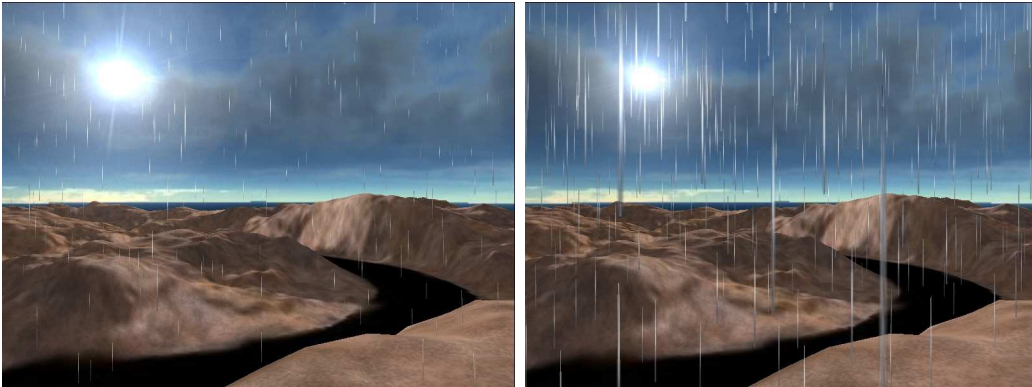


Fig. 14. Using both extensions simultaneously, with 4000 particles. Left: light rain, short and narrow streaks. Right: heavy rain, long and large streaks.

convincing than the image on the left, generated without the extension.

The left image of Figure 13 presents the illumination extension used with larger drops. The right image shows raindrops illuminated by two colored light sources simultaneously. We can observe that the drops on the left of the

image appear more yellowish and those on the right appear more reddish.

Figure 14 illustrates the compliance of our extensions, presenting streaks illuminated from a white light source.

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>.

9 Conclusions 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. Neither of these methods can 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 visually realistic real-time high frame-rate execution.

In future works, we will 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 partially 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.
- [Bir03] V. Biri. *Techniques d’animation dans les méthodes globales d’illumination*. PhD thesis, Université de Marne La Vallée, 2003.
- [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.
- [Gla94] A. S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.
- [GN03] K. Garg and S. K. Nayar. Photometric model of a rain drop. Technical report, Columbia University, 2003.
- [GN04] K. Garg and S. K. Nayar. Detection and removal of rain from videos. In *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004), with CD-ROM, 27 June - 2 July 2004, Washington, DC, USA*, volume 1, pages 528–535, 2004.
- [Gre75] A. W. Green. An approximation for the shapes of large raindrops. *J. Appl. Meteor.*, 21:1578–1583, 1975.
- [GS04] S. Guy and C. Soler. Graphics gems revisited: fast and physically-based rendering of gemstones. *ACM Trans. Graph.*, 23(3):231–238, 2004.
- [Har03] M. J. Harris. *Real-Time Cloud Simulation and Rendering*. PhD thesis, University of North Carolina, 2003. Technical Report #TR03-040.
- [Igl04] A. Iglesias. Computer graphics for water modeling and rendering: a survey. *Future Gener. Comput. Syst.*, 20(8):1355–1374, 2004.
- [KIY99] K. Kaneda, S. Ikeda, and H. 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.
- [KSW04] P. Kipfer, M. Segal, and R. Westermann. Uberflow: A GPU-based particle engine. In *Eurographics Symposium Proceedings Graphics Hardware 2004*, pages 115–122, 2004.

- [LZK⁺04] M. S. Langer, L. Zhang, A. W. Klein, A. Bhatia, J. Pereira, and D. Rekhi. A spectral-particle hybrid method for rendering falling snow. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering)*. ACM Press, june 2004.
- [MS95] P. W. C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *Symposium on Interactive 3D Graphics*, pages 95–102, 211, 1995.
- [PA00] S. Premoze and M. Ashikhmin. Rendering natural waters. In *PG '00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, page 23, Washington, DC, USA, 2000. IEEE Computer Society.
- [Ros00] O. N. Ross. Optical remote sensing of rainfall micro-structures. Master’s thesis, Freie Universität Berlin, 2000. in partnership with University of Auckland.
- [SDY02] T. Sato, Y. Dobashi, and T. 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.
- [SFD00] Y. Sun, F. Fracchia, and M. Drew. Rendering diamonds. In *11th Western Computer Graphics Symposium (WCGS)*, pages 9–15, 2000.
- [SKALP05] L. Szirmay-Kalos, B. Aszodi, I. Lazanyi, and M. Premecz. Approximate ray-tracing on the gpu with distance impostors. In *Computer Graphics Forum*, volume 24, issue 3, pages 685–704, 2005.
- [SRNN05] B. Sun, R. Ramamoorthi, S. G. Narasimhan, and S. K. Nayar. A practical analytic single scattering model for real time rendering. *ACM Trans. Graph.*, 24(3):1040–1049, 2005.
- [SW03] S. Starik and M. Werman. Simulation of rain in videos. In *3rd international workshop on texture analysis and synthesis (Texture03)*, pages 95–100, Nice, France, 2003.
- [Tho01] S. Thon. *Representation de l’eau en synthèse d’images. Exemple de grandes étendues d’eau*. PhD thesis, Université de Limoges, 2001.
- [WMT05] H. Wang, P. J. Mucha, and G. Turk. Water drops on surfaces. *ACM Trans. Graph.*, 24(3):921–929, 2005.
- [WW04] N. Wang and B. Wade. Rendering falling rain and snow. In *ACM SIGGRAPH 2004 Technical Sketches Program*, 2004.
- [Wym05] C. Wyman. An approximate image-space approach for interactive refraction. *ACM Trans. Graph.*, 24(3):1050–1053, 2005.
- [YZZ04] Y. Yang, C. Zhu, and H. Zhang. Real-time simulation: Water droplets on glass windows. *Computing in Science and Eng.*, 6(4):69–73, 2004.