Eurographics Workshop on Natural Phenomena (2006)
E. Galin, N. Chiba (Editors)

Artist-Directable Real-Time Rain Rendering in City Environments

Natalya Tatarchuk[†] and John Isidoro[‡]

ATI Research

Figure 1. *Photorealistic rain rendering using our system on the left versus the same complex scene without the rain. Note the multitude of effects in the rainy environment, including rainfall rendering, dripping raindrops, glow, streaky reflections and such details as the tire treads in the street puddles.*

Abstract

Photorealistic rain greatly enhances the scenes of outdoor reality, with applications including computer games and motion pictures. Rain is a complex atmospheric natural phenomenon. It consists of numerous interacting visual effects. We present a comprehensive real-time system for the realistic rendering of rain effects in complex environments in real-time. Our system is intuitive, flexible and provides a high degree of artistic control for achieving the desired look. We describe a number of novel GPU-based algorithms for rendering the individual components of rain effects, such as a hybrid system of an image-space approach for rainfall and the particle-based effects for dripping raindrops and splashes; water surface simulation for ripples; animation and rendering of water droplets trickling down on transparent glass panes; view-dependent warped reflections and a number of additional effects. All our techniques respond dynamically and correctly to the environment lighting and viewpoint changes as well as the atmospheric illumination due to lightning. Our effects can be rendered at interactive rates on consumer graphics hardware and can be easily integrated into existing game and interactive application pipelines or offline rendering.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. Introduction

Creating a faithful representation of rain in complex natural environments is a non-trivial problem. The challenge

[†] natasha@ati.com
[‡] jisidoro@ati.com

stems not only from the visual complexity and diversity of the rain components and scene objects, but also from the huge amount of small details that should be modelled to obtain realistic visual effects and physically plausible simulation. However, rain rendering greatly enhances outdoor scenes and is an important problem for computer graphics, with many applications in computer games and motion pictures. Filming rain scenes involves a significant effort and cost due to complicated setup. This task becomes even more challenging when trying to create photorealistic rendering of rain in rich environments at interactive rates.

Some recent games which incorporate rain rendering use simplistic approaches, including rendering stretched, blended particles to simulate falling raindrops or using blended animated textures (as in [WW04]) to render precipitation. These methods fail to create a truly convincing and interesting rain impression. Furthermore, games often limit using only one or two individual rain effects (the rain particles or the scrolling textures and perhaps a CPU-based water puddle rendering) to simulate the impression of rainy environment. This results in an unrealistic rendering with the rain not reacting accurately to scene illumination, such as lightning or spotlights.

Rain is an extremely complex atmospheric natural phenomenon. It consists of numerous visual effects interacting together. Although research has been done in the area of rendering some individual components (rain streaks in [GN06], [WW04], or water droplets on surfaces in [KIY99], [WMT05]), there exists a gap for creating a complete system for rain rendering in complex environments. Simply adding several individual components is insufficient as the discerning viewer's eye quickly notices the missing elements. For a truly convincing illusion of rain environment we must present a coherent system supporting the full gamut of the natural phenomena associated with rain.

There is a strong need for inexpensive and streamlined algorithms capable of photorealistic rendering of rainfall and rain-related effects in games and interactive applications. Photorealistic rain rendering requires convincing display of rainfall and raindrops, various dynamic water-related effects for puddles and streaming water, and a variety of scene effects for atmospheric effects and wet materials. Our main contribution is an intuitive, comprehensive and flexible system for photorealistic rendering of rain effects in real-time in a complex environment. We provide a high degree of artistic control for achieving the desired final look. To our knowledge, this is the first complete system of this kind. We present a number of novel algorithms for rendering the individual components of rain, including the following:

- A new post-processing composite rainfall algorithm exhibiting raindrop shape perturbations and dynamic response to varied illumination conditions and viewpoints
- Simulation and rendering of raindrops dripping from various objects in the scene
- Techniques for raindrop splashes and splatters on solid objects and in water puddles
- An engine-driven lightning illumination system for simulating lightning flashes
- Halos around light sources and objects due to light scattering in rainy mist
- A novel effect for rendering view-dependent warped reflections on wet surface materials and puddles using reflection impostors
- Atmospheric light attenuation
- GPU-based water surface simulation for puddle ripples due to raindrop splashes
- A novel approach for the simulation and rendering of water droplets on glass surfaces on the GPU, with wetting, droplet merging and separation phenomena
- A large number of supporting effects resulting in increased scene realism

Our algorithms provide a variety of artist-directable controls and respect the rules of physics for simulating rainfall. All our techniques utilize a unified HDR illumination model to allow the rain to respond dynamically and correctly to the environment lighting and viewpoint changes as well as the atmospheric effects (such as lightning). The illumination system also provides integrated support for dynamic soft shadows. Our effects can be rendered at interactive rates on consumer graphics hardware and can be easily integrated into existing game and interactive application pipelines or offline rendering to enhance the scene realism. We have tested our system extensively and successfully in a complex environment, representative of future-generation games.

2. Related work

Rain effects have been examined in the context of atmospheric sciences ([WC75] and [Mas75]), as well as in the field of computer vision ([NN03], [GN04]). However, at the moment only a few approaches exist for creating realistic rainfall rendering that dynamically responds to the lighting environment and camera movement.

Constant brightness rain strokes are generated in [SW03] for simulation of rain in videos. This approach fails to represent dynamic illumination and camera movement. In [WW04] rain and snow precipitation was modelled with several interpolated hand-drawn textures with constant brightness mapped on a double-cone which is dynamically aligned to match the camera orientation in real-time. A detailed photometric variation model for modelling rain streak illumination is described in [GN06], accounting for raindrop oscillations and motion parallax. Rainfall is rendered with a particle system using a large precomputed rain streak texture database to interpolate the streak appearance. Although this offline approach exhibits dynamic response to varying lighting conditions and camera movement, it does not consider the illumination due to lightning and the effects of atmospheric scattering and light attenuation.

Cinematographic rendering of rain uses various approaches to create visually appealing results. In *The Matrix Revolutions* ([BSS*04]), instanced 3D rod-like shapes for water drops and hand-animated raindrop splash objects were used for rendering rain effects. Commercial software such as Maya(R) and 3D Studio Max(R) include offline systems for rain rendering based on particle systems. These methods generate detailed physics-based movement and accurate visual representation at the price of prohibitive computational speeds. In Pixar's *A Bug's Life*, probability distribution functions are used to guide the fluid simulation of stylized raindrop splashes ([Her01]). The raindrops were rendered with a particle system, using implicit surfaces to represent individual particles. This is a computationally expensive approach and thus poses difficulties in interactive applications.

Atmospheric light scattering in the context of rain precipitation due to weather conditions is analyzed in [NN03]. The authors show that the appearance of glows around the light sources is due to multiple scattering effects. Scattering effects due to clouds and atmospheric particles illuminated by lightning were described in [YYT01] and [ENN90]. Illumination due to lightning is important for the realism of inclement weather rendering. In the scope of our system, we only focus on approximating the atmospheric scattering and modelling the illumination due to lightning flashes for the scene.

In the field of computer graphics there exists a plethora of approaches for fluid simulation and rendering, based on computational fluid dynamics ([FM96], [Sta99], [Har03]). However, due to high computational complexity these approaches are less practical for most interactive applications. Many applications use CPU-based water displacement via a dynamically displaced vertex mesh (as in [Gom00]).

Animation of water droplets with the use of particle systems in a discrete environment driving the droplet movement was described in [KIY99] using an offline implementation. Environment mapping was used to model droplet reflection and transparency. Discrete surface representation allowed easy integration of obstacles for droplet movement simulation (such as windshield wipers) and provided support for such effects as merging and wetting. Flowing water droplets movement is modelled in [FHP99] with a mass-spring system simulating surface tension and volume conservation constraints. An offline physically-based method for droplet simulation is presented in [WMT05], using level set distance field representing the surface of the water droplet to simulate a variety of physically accurate small-scale fluid phenomena.

The remainder of this paper is organized as follows. Rain precipitation effects and raindrop splashes are described in section 3. Water puddles and droplet simulation and rendering are covered in sections 4 and 5. We present the details of scene rendering effects, such as lightning illumination, reflections, and atmospheric effects in section 6. The results and conclusions are given in sections 7 and 8.

Figure 2: *Rendering rainfall and raindrops with splashes*

3. Rendering rain precipitation

Precipitation due to rain consists of spatially distributed water drops falling at high velocity, refracting and reflecting the environment around it. As the raindrops fall through the scene, they create the perception of motion blur and parallax and generate ripples and splashes in puddles. We developed a hybrid system of an image-space approach for the rainfall and particle-based effects for dripping raindrops and splashes. We render individual raindrop shape variation and motion parallax due to different depth for raindrop movement as well as dynamic raindrop illumination. Unlike purely particle-based approaches, the image-based rainfall precipitation effect does not incur extra performance overhead for modelling heavy versus light precipitation.

3.1. Rendering multiple layers of rain with a post-processing composite effect

Our image-space rainfall effect simulates multiple layers of falling raindrops in a single compositing pass over the rendered scene. This method differs from most previous approaches in rendering the rainfall without the use of a particle system with a large number of particles or rain textures. We provide a set of artist controls for the rain direction, velocity, and strength. The raindrop rendering receives dynamically-updated parameters such as the lightning brightness and direction from the lightning system to allow correct illumination resulting from lightning strikes.

Creating rainfall with multiple layers of rain. Com-

puter vision analysis of rain models ([GN04]) and video rain synthesis ([SW03]) shows that one cannot easily recognize rainfall from a single static frame. However, rain is easily noticeable in a dynamic simulation or a video. Perceptual analysis of rain video shows that the individual raindrop motion cannot be tracked by human perception accurately due to swift movement and density of raindrops. This allows us to assume temporal independence of rain frames. However, our empirical experiments showed that purely random movement of raindrops does not yield satisfactory results and generates excessive visual noise. Therefore to simulate strong rainfall, we simultaneously use the concepts of individual raindrop rendering and the principles of stochastic distribution for simulation of dynamic textures (as in [DCWS03]).

We render a composite rainfall layer prior to the final post-processing of the rendered scene. We must consider the practical performance implications of the rainfall layer as a full-screen pass and design the algorithm to yield pleasing visual results without expensive computations.

The first challenge lies in minimizing the repeating patterns that are inevitable when using a single static texture to model dynamic textured patterns. Initial raindrop distribution in the full-screen pass is simulated with an animated 8 bit raindrop placement texture. Artists can specify the rain direction and speed in world-space to simulate varied rainfall strength. At every time step we determine the raindrop clip space position (x_i, y_i) for every pixel in the composite pass. Using an artist-specified rain direction vector v_r in clip space, the current raindrop position, and the rain speed, $|v_r|$ we compute the tentative raindrop distribution texture coordinates as follows $(x_i, y_i) = v_r^{cp} * |v_r| * \triangle t$.

In order to create the illusion of several layers of raindrops, the artists specify a rain parallax parameter p_r which maps the depth range for the rain layers in our scene. Using the concepts of stochastic distribution for simulation of dynamic textures, we compute a randomized value for an individual raindrop during the simulation, r_i. Using the rain parallax value p_r, the screen-space individual raindrop location (x_i, y_i) for a given pixel computed earlier and the distribution parameter r_i, we can model the multiple layers of rain in a single pass with a single texture fetch. The parallax value for the raindrop, multiplied by a distribution value, is used as the w parameter for a projective texture fetch to sample from the rainfall movement texture: $w_i = p_r * r_i$. This allows us to simulate raindrops falling with different speeds at different layers of rain without obvious repeating patterns.

Rain appearance. Raindrops refract light from a large solid angle of the environment (including the sky) towards the camera. Specular and internal reflections further add to the brightness of the drop. Thus, a drop tends to be much brighter than the portion of the scene it occludes. The solid angle of the background occluded by a drop is far less than the total field of view of the drop itself. In spite of being transparent, the average brightness within a stationary drop (without motion-blur) does not depend strongly on its background.

The illumination for rain precipitation is computed using water-air refraction for individual raindrops as well as reflection due to the surrounding light sources and the Fresnel effect. The layer of rain is shaded by using a normal map of varied individual raindrop shapes. Our approach does not require any preprocessing and can handle an arbitrary number of light sources. The lighting model for illuminating individual raindrops is flexible. Currently our system utilizes point light sources.

Although we render the entire 'curtain' of rainfall in a single pass, we model the raindrop shape variation phenomenologically. The normal for a given raindrop pixel n_i is distorted as function of the raindrop velocity v_r and w_i (as computed above), thus mimicking the complex shape deformations that the raindrops undergo as they fall through the air during rain. We compute an individual raindrop shape distortion parameter using the raindrop velocity and position values, the distribution of raindrops (as described above) and the parallax parameter along with the scene depth information. This distortion parameter is used as an LOD bias for the raindrop normal map fetch, thus serving to create additional motion blur.

This, combined with a fully dynamic lighting, gives us the desired variation and interaction for the raindrop shapes and illumination. To capture the complex interactions between the raindrops and the scene light sources, the viewer and the rendered scene, we compute the specular illumination and reflection based on the individual raindrop normal and air-to-water refraction. These contributions are attenuated toward the edges of the raindrop by using a variation of the Fresnel equations with the following formula: $f_i = 0.95 \cdot (1.0 - \overrightarrow{n_i} \cdot \overrightarrow{v_i})^4 + 0.05$, where $\overrightarrow{n_i}$ is the raindrop per-pixel normal, and $\overrightarrow{v_i}$ is the per-pixel view vector.

We note that falling raindrops produce motion-blurred intensities due to the finite shutter speed of a camera. Unlike a stationary drop, the intensities of a rain streak depend on the brightness of the drop as well as the background scene radiances and integration time of the camera. We simulate the motion blur and the strong mistiness of the falling raindrops by applying blurring via post-processing (as described in section 6.2) after the rain pass has been blended onto the scene rendering. The amount of blurring is controlled by the raindrop velocity and the input rain opacity texture. This simulates both raindrop motion-blur and multiple-scattering glow for individual raindrops, taking into account the raindrop environment.

Rain and lightning flashes. As lightning strikes, the raindrops should appear more transparent, reflective and refractive. Thus the opacity of each individual raindrop must be a function of the lightning brightness (see Figure 3.1); otherwise water surfaces appear too solid. Our rendering script propagates the lightning system parameters (see section 6.1)

(a) (b)

Figure 3: *Rendering the raindrops falling off the rooftop ledge.*

to all of our rain shaders, as well as to the material shaders. For the raindrop rendering, we use a combined lightning brightness parameter (mixing both lightning 'light sources' as they flash in the environment) to compute the bias value to adjust the amount of reflection and refraction and the raindrop transparency as follows: $l_b = 1.0 + k_c \cdot (L_1 + L_2)$ where l_b is the lightning bias for computing the resulting raindrop illumination, k_c is the artist-specified rain contrast value (depending on the rainfall strength), and L_1, L_2 are the lightning brightness values for two lightning illumination sources. The resulting value l_b is used to modulate the reflection and refraction terms as well as the specular illumination and the opacity of the raindrop.

Other practical considerations. Realistic rain is very faint in bright regions but tends to appear stronger when the light falls in a dark area. Physically accurate modelling results in overly dim rain appearance. We use a cinematic technique of adding milk to water while filming rain as inspiration and bias the raindrops color toward the white spectrum to create a stronger perception of rainfall.

3.2. Rendering dripping raindrops

During rain, raindrops drizzle from various objects in the scene - trickling off gutter pipes, window ledges and so on (see Figures 3 and 3.1 for some examples). We simulate this effect with the use of physics-based particle systems using screen-aligned billboard representation for individual raindrops. The base particle system simulation uses the physical forces of gravity, wind and several animation parameters for raindrop movement. The artists can place any number of separate particle systems, culled by the camera frustum during rendering, throughout the environment to generate dripping raindrops.

Each raindrop has its initial shape specified via a droplet normal map. We model the raindrop shape variation by stochastic motion-based shape elongation. A pre-blurred and a priori stretched normal map helps in increasing the perception of motion blur. We use raindrop depth and velocity to drive shape elongation and dynamic blurring of the normal map, and therefore resulting drop illumination (by using LOD bias for normal map lookup). We compute specular reflection and air-to-water refraction effects for each individual raindrop in the same manner as in the rainfall method.

To control raindrop transparency, we attenuate raindrop opacity by its distance in the scene. We wish to make the individual raindrop particles appear less solid and billboard-like as they move through the environment. This can be accomplished by attenuating the particle opacity value by the Fresnel value (same as the composite rainfall) f, scaled and biased by two artist-specified parameters for droplet edge strength e_s and bias e_b(which could be specified per particle system): $\alpha' = d_p \cdot \alpha \cdot (e_s \cdot f + e_b) \cdot (1 - \frac{l_b}{2})$ where d_p is the particle distance and l_b is computed in the same fashion as in Section 3.1, α is the initial texture-based raindrop opacity. We used the observation that the raindrops should appear more transparent and water-like when the lightning strikes, and increased the raindrop transparency as a function of the lightning brightness to maintain physical illusion of water. This can be easily done by biasing droplet transparency by $1 - \frac{1}{2} \cdot l_b$. The particles still maintain their artist-specified transparency in the regular lighting without any lightning flashes. We used this approach for both regular raindrop rendering and for raindrop splash rendering.

3.3. Raindrop splashes

We simulate raindrops splashing when hitting solid objects by colliding individual particles with objects in the scene (Figures 3.1, 3 and 3.3). A single filmed high-quality milk drop splash sequence (Figure 4) is used to drive rendering of the thousands of raindrop splashes. We incorporate a high degree of randomization for splash particle parameters (such as size and transparency) in order to reduce the noticeable visual repetition of the splash animation. Furthermore, we

randomly flip the horizontal texture coordinates based on particle parameter. Splashes should appear correctly lit by

Figure 4: *Milk drop sequence for raindrop splash animation.*

the environment lights. Empirical observations show that the splashes appear the strongest when backlit and thus display the subtle effects of raindrops splashing under a street light. Thus if light sources are behind the rain splashes, we render the splash particles as brightened backlit objects; otherwise we only use ambient and specular lighting for simplicity. . Finally, we integrate the illumination from the lightning flashes as described in Section 6.1 for the raindrop splash illumination. Figure 3.3 shows two examples of the lit splashes. Aside from the dynamic lights, we wanted to

(a) Regular rendering (b) During a lightning flash

Figure 5: *Raindrop splash rendering*

simulate the splashes lit by all of the bright objects in the environment (such as street lamps, for example), even though those objects are not actual light sources in our system. Using an overhead illumination texture we can simulate the sky and street lamp source lighting. The light from these objects is encoded into this texture at preprocessing stage. During the splash rendering we can use the world-space position of each splash particle to sample the overhead illumination information and use it to modulate splash brightness.

3.4. Misty object halos due to precipitation

In a strong rainfall, as the raindrops strike solid objects, they generate not only the splashes, but also the delicate halo outlines along the edges of objects. This is a very important visual cue which has been omitted from most of the existing rendered rain environments. We support rendering of this effect for objects in our scene (including the animated objects, such as cars, see Figure 6) by using normal 'fins' (similar to fur rendering in real-time in [LPFH01]). To create a rain halo effect, we insert a degenerate quad which is extruded normal to the surface at object silhouettes. The actual halo is rendered on each such quad by using the rainfall algorithm from section 3.1 as an animated texture, alpha-blended with the rest of the environment.

Along with the water splashes from fast and heavy raindrops, strong rainfall also generates a more subtle effect with the raindrop splattering on the surface of wet materials (Figure 6). We use a shells-based technique to create the raindrop splatters. The shells technique is widely used for rendering fur in real-time (as described in [LPFH01]). We render the material with raindrop splatters as a series of extruded shells around the original object. The rain splatters are rendered on the surface of objects in the form of concentric circles. In each successive shell we expand the splash circle footprint with a series of animated texture fetches and blend onto the previous shells. This creates a very convincing effect of dynamic splatters on objects due to raindrops.

4. GPU-Based Water Simulation for Puddle Rendering

The raindrop particle collisions generate ripples in rain puddles in our scene. The goal was to render dynamic realistic wave motion of interacting ripples over the water surface using the GPU for fast simulation. We use an explicit integration scheme to simulate fluid dynamics for rendering dynamically lit puddle ripples. Similar to real-life raindrops, a single raindrop in our system excites multiple interacting ripples on the water surface. The physics simulation for water movement is done entirely on the GPU. We treat the water surface as a thin elastic membrane, computing forces due to surface tension and displacing water sections based on the pressure exerted from the neighboring sections. Our system provides simple controls to the artists to specify water puddle placement and depth. Figure 7 shows water puddle on a rooftop.

Water surface displacement computation. Water ripples are generated as a result of raindrops falling onto the geometry in the scene. Our system supports generation of raindrop ripples as a result of direct collision as well as due to a randomized spatio-temporal distribution. The latter is accomplished by seeding the water ripple texture with raindrop masses at rendering time.

We approximate the water surface as a lattice of points on

Figure 6: *Object halos due to rain precipitation and raindrop splatters on the car surface. Note the close-up at the bottom left corner.*

Figure 7: *Dynamic water simulation for puddle rendering. Note the view-dependent reflections*

the GPU containing the information about the water surface in that location (we store the current and previous time step wave displacement values). These quantities can be packed into a single 32 bit texture using 16 bit per channel, giving a good precision balance for computing displacements.

Due to memory considerations, we currently use the stochastic seeding method, rather than direct collision response, for a simulation on a 256x256 lattice. We splatter the raindrops as point primitives into the water simulation texture with the RGB value proportional to the raindrop mass during the first pass of the simulation. This method can be applied to generate dynamic water surface response for arbitrary objects. This can be achieved by rendering an orthographic projection of the objects into the seeding texture using the object's mass as the function for color of the object's outline. This would generate a wake effect in the water surface.

The rendered seeds act as the initial ripple positions by exciting the ripple propagation in the subsequent passes. Real-life raindrops generate multiple ripples that interact with other ripples on the water surface. We implement the same model. We render a raindrop into a wave seed texture using a dampened sine wave as the function for raindrop mass. This approximates the concentric circular ripples generated by a typical raindrop in the water puddle.

To compute the water surface response we treat it as a thin elastic membrane. The forces of gravity are considered negligible for the purposes of the simulation (as compared to the surface tension). At every time step, infinitesimal sections of the water surface are displaced due to tension exerted from their direct neighbors acting as spring forces to minimize space between them. Vertical height of each water surface point in a cell *(i, j)* can be computed with partial differential equation:

$$\frac{\partial^2 z_{i,j}}{\partial t^2} = v_{i,j}^2 \left(\frac{\partial^2 z_{i,j}}{\partial x_{i,j}^2} + \frac{\partial^2 z_{i,j}}{\partial y_{i,j}^2} \right)$$

where $z_{i,j}$ is the water displacement height, $v_{i,j}$ is the velocity of the water in the cell, $x_{i,j}$ and $y_{i,j}$ are the lattice coordinates.

We use explicit Euler integration in DirectX9.0 pixel shaders to solve this PDE in real-time by using a texture feedback approach to determine the water wave heights for each point on the lattice. We found that two passes are sufficient for a stable simulation. During the final pass we compute the normals for the water displacements using the Sobel filter.

Water puddles integration.. We sample from the water membrane simulation using the object's current position in world space (the xz Cartesian coordinates) as a lookup texture coordinates into the computed ripple wave normal map. Since our system implements a single ripple simulation for all puddle surfaces due to memory considerations, this limitation is overcome by providing the artists control over the ripple sampling space. To reduce visual repetitions of the resulting puddles, we provide a per-object scale parameter s_o for ripple waves and a rotational angle θ_o for the ripples look-up. The ripple simulation sample coordinates are rotated in texture space based on the specified object angle θ_o. Note that no additional geometry is required for puddle integration. This approach also enable our system to control turning on and off of puddle rendering on demand by using a material parameter and dynamic flow control features of the latest shader models.

To render an object with water puddles, we perturb the original object's bump map normal with the normal from the water membrane simulation. The artists can also specify a puddle ripple influence parameter per object. This parameter controls the perturbation ratio for the water ripple normal and the original bump map normal, allowing creation of different water motion for various objects.

Puddle Placement and Depth. In real environments, water puddle depth and locations differ significantly due to landscape details and rainfall accumulation. Our system provides complete artistic control over the puddle placement and depth with a puddle depth mask. This mask specifies both the location of each puddle in the environment and its depth variation. Adding puddles with dynamic ripples to objects is intuitive with this approach. During rendering, we first sample the puddle depth map for the current depth value d_i. Then the ripple normal map is sampled as described earlier. We observe that the deep puddles' visual properties depend mainly on the color of the underlying material (for example, the asphalt on the street), and the water surface geometric properties for illumination. As the light rays refract through the water surface, the viewer observes the color properties of the material. However, the actual micro-geometric structure of the surface under the puddle does not influence the appearance of the puddle. Therefore to modify the apparent puddle depth, we can specify the influence of the water surface normal as compared to the object normal vector p_i. We interpolate between the object nor-

mal vector and the water surface normal based on d_i and an artist-specified puddle influence parameter p_i. Using this perturbed normal, we render the objects with water surfaces using Fresnel equations ([Jen01]) for water-air refraction and reflection, as well as the material properties of the object as desired.

5. Water droplet animation and rendering on glass surfaces in real-time

We adopted an offline raindrop simulation system from [KIY99] to the GPU to dynamically animate and render a large number of water droplets and their streams trickling down on glass planes in real-time (Figure 5). We animate and render the droplets entirely on the GPU, with the simulation using the gather operation in the pixel shader, rather than the original scatter-based particle system implementation. The shape and motion of water droplets is influenced by the forces of gravity and the interfacial tension force, as well as air resistance. We generate the quasi-random meandering of raindrops due to surface tension and the wetting of the glass surfaces due to water trails left by droplets travelling on the surface. Our system produces correctly lit droplet appearance including the refraction and reflection effects.

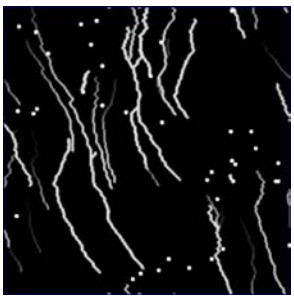
Figure 8: *Water droplet rendering on the glass window*

Droplet movement. We represent the glass surface as a discrete lattice of cells, storing the water mass $M_{i,j}$ at that location, the velocity $v_{i,j}$, and the droplet traversal quantity $t_{i,j}$ within each cell *(i, j)*. The droplet information is packed into a 16-bit per channel RGBα texture. The droplet begins to trickle down the glass surface when the acting downward forces start to exceed the upward resisting forces on the droplet. Droplet movement direction is determined by external forces acting on the droplet, however, the meandering of the droplet path also depends on the surface properties of the glass (due to impurities, small scratches or grooves). Additionally we can account for obstacles on the droplet path which can be encoded into the cell information. Each lattice cell stores the affinity parameter $k_a(i,j)$ (artist-specified or assigned at random from a normal distribution) which describes the hydrophobic or hydrophilic properties of that surface location.

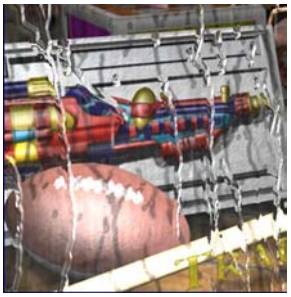
We compute the force of gravity F_g at render time as a function of the droplet mass: $F_g(i,j) = M_{i,j} \cdot g$ (where g is the gravitational acceleration) and use it as the downward force $F_d(i,j)$. For the competing upward forces, the static friction force $F_s(i,j)$ is specified for stationary droplets and we compute the dynamic friction for moving droplets, $F_{df}(i,j)$. The glass surface friction coefficients $k_f(i,j)$ are specified via a texture. These forces and the surface tension force $F_t(i,j)$ vary over the surface of the glass based on the affinity parameter $k_a(i,j)$. At every step of the simulation, we apply the resultant force, $F_{i,j}$ to the current droplet velocity to compute the new velocity value for the droplet: $v'_{i,j} = v_{i,j} + \frac{F_{i,j}}{M_{i,j}} * \triangle t$ where $\triangle t$ is the current time step. We use texture feedback technique for a stable simulation.

Since we limit our simulation to the downward movement on glass surfaces, at every time step a droplet can only flow into the three neighbor cells directly below the current cell(as opposed to eight in [KIY99]). The new cell for the flow is randomly chosen. The probability depends on the droplet velocity vector, the affinity of the current cell and the 'wetness' of the target cells. We compute the probability value for each droplet cell using the roulette areas approach as described in [KIY99]. Droplet flow has greater affinity toward the wet regions of the surface. During the droplet traversal, some amount of water must remain behind. Mass transfer from the current cells allows us to simulate this wetting phenomenon for the droplet movement. We support droplet merging in the following manner - if any droplets arrive at the same cell, we add their mass values and maintain a single droplet thereon. At the end of each simulation step, we compute the droplet velocity $v'_{i,j}$, new droplet mass value $M'_{i,j}$ and the normal vector $n_{i,j}$ based on this mass for each water cell *(i, j)* on the lattice.

Droplet rendering. The background scene is rendered prior to the droplet rendering pass. Water density values are treated as height values. We apply a Sobel-type filter to derive per-pixel normals based on water density values for the droplets. We use the computed droplet normals $n_{i,j}$ to perturb the background scene to simulate reflection and refraction through the water droplets on the glass surface (Figure 9(b)). Droplet water density value is used as the environment reflection and refraction coefficient parameter to control the amount of environment refraction and reflection through the droplets. The droplets refract the background scene (sampled from an offscreen texture) and the environment in front of the glass window (by sampling the planar reflections offscreen buffer). Aliasing reduction is achieved with applying a Fresnel term and edge attenuation to the droplet reflections and refraction (as described in section 3.1). Note that for water droplets we determine the droplet edge for each pixel and scale the exponent used to compute Fresnel term in order to



(a) Raindrop mass values



(b) Droplet shadows on surrounding objects

Figure 9: *Water droplets refracting the background scene and reflecting external lights. Note the bright quasi-caustic highlight for heavy droplets*

reduce aliasing at droplet edges. Droplet shadows on surrounding objects are rendered by using the droplet mass values as projective textures (Figure 9(a)). If the droplet mass value in a cell is above a artist-specified threshold, we render a quasi-caustic highlight in the middle of the projected shadow for that droplet.

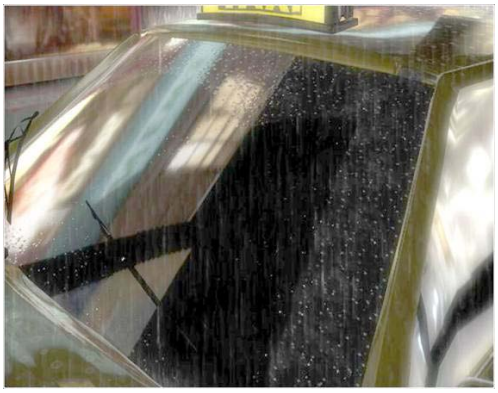


Figure 10: *The windshield wipers can dynamically wipe away the raindrops on the glass surface*

6. Scene rendering effects

The realism of an outdoor rain rendering depends on the multitude of details present in such an environment as much as on the convincing rendering of the rain and water effects. The illumination of the environment, the lightning flashes, the atmospheric light scattering effects - all of these help enhance the resulting rendering. Wet environments also display a great deal of reflectivity - without realistic reflections the illusion is broken.

6.1. Lightning system and integration

A dark night in rough weather would not affect the viewer in the same manner without the sudden surprise of a lightning flash followed by the inevitable thunder. Lightning is a strong directional light that affects every object in the scene. Creating a realistic lightning effect in interactive applications is challenging for several reasons. Illumination from the lightning flashes needs to simultaneously affect every object in the scene. Uniformly aligned shadows are crucial. Simply adding extra shadowing lights for each lightning is still an impractical approach for interactive applications due to associated performance cost and additional memory requirements for storing shadow maps.

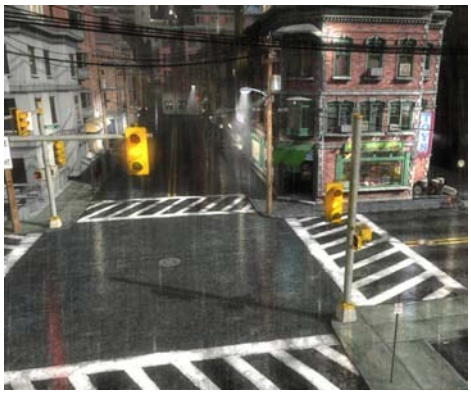


Figure 11: *Illumination due to a lightning flash - note the uniformly aligned shadows due lightning illumination*

Our proposed solution consists of a system driving lightning flashes and the resulting illumination model consistently integrated into all materials and effects. We support several simultaneous lightning flash light sources. During the preprocessing phase for the environment several key lightning source directions are picked by the artists and global illumination illumination solutions are computed for each selected light source. The encoded illumination value (in a series of 8 bit textures per direction) is used at rendering time by all rendering components to modulate the illumination result to account for a lightning flash event. This compact representation allows us to create consistent uniform lightning shadows (as in Figure 11).

In order to control the lightning flash sequence at runtime, the engine uses a scripting language to create a mix of the cardinal lightning directions which are encoded in the above textures. We provide the artists with an animated editable intensity parameter for the mixing of the cardinal lightning directions. The rendering script provides a method for computing overall lightning brightness value at every frame.

Every shaded pixel in our environment uses the lightning illumination information. The computed lightning direction and brightness parameters are used at render-time with the encoded lightning illumination information to create accurate lighting result for each material or effect (as described in section 3). The rendering script propagates the animation parameter for each of the two lightning flashes to all of the shaders in the form of uniform parameters. The lightning illumination value is added to the regular illumination for each material prior to tone mapping. This approach incurs negligible performance cost (consisting of a single texture fetch along with several ALU operations to compute the result of several lightning flashes). All objects in our real-time environment use this scheme and thus appear to respond accurately to lightning illumination in the scene.

6.2. Post-processing system for rendering atmospheric effects due to light scattering and rain precipitation

In recent years post-processing has become a popular approach for adding visual variety to games, as well as for approximating many camera or environment properties of the world around us. We used a flexible post-processing pipeline to simulate atmospheric effects such as misty glow due to light scattering, to perform tone mapping for HDR rendering and for a variety of specific blurring effects for creation of rain effects.

Creating the appearance of glow due to inclement weather. Water particles in the atmosphere during the rain increase the amount of light scattering around objects. Multiple scattering effects are responsible for the appearance of glow around light sources in stormy weather ([dH57] and [NN03]). In order to approximate the effect of halos around bright light sources, we make use of the post-processing pipeline that is controllable through a rendering script.

To approximate the atmospheric point spread function which can be used to model multiple scattering around the light sources in the stormy weather, we use the Kawase post-processing approach for rendering glows in our scene ([Kaw03]). To model fog attenuation due to water scattered in the atmosphere we implemented light attenuation based on distance in material shaders. We attenuate the light information based on distance directly in the shaders. In the vertex shader we compute the distance of the object to the observer and then compute the linear fog value which is then sent to the interpolator for rasterization. See Figure 12 for an example of both effects.

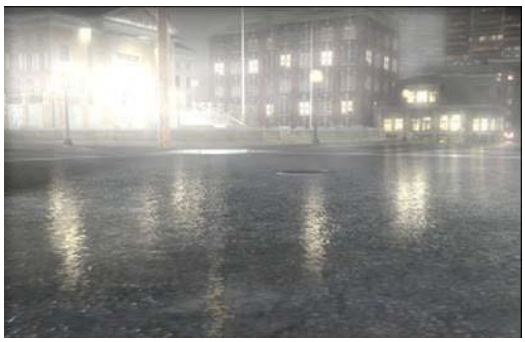


Figure 12: *Atmospheric light scattering and attenuation*

6.3. Reflections

Realistic streaky reflections increase the feel of rain on wet streets and various object surfaces. Therefore, adding convincing reflections is a must for any rainy environment. To simulate the appearance of a wet city street in the rainy night, we render a number of various reflection effects in our scene (as seen in Figures 13, 11):

- View-dependent stretched warped reflections on surfaces
- Wet material rendering with environmental reflections
- Planar reflections on glass surfaces
- Dynamic cubemap environment reflections for animated objects, such as reflections on the surface of a moving vehicle

Depending on the polygonal properties of a particular object, highly specular surfaces can display a great deal of aliasing if one is not careful. The solution lies in attenuating the reflection illumination and specular highlights at the objects' edges. The artists can supply a per-object coefficient f_o for computing Fresnel effect as follows: $f = (1 - \overrightarrow{n} \cdot \overrightarrow{v})^{f_o}$ where $\overrightarrow{n}$ is the per-pixel normal and $\overrightarrow{v}$ is the view vector for a given pixel. Using this value we can then compute the attenuation coefficient for the reflective materials using the following formula: $k_a = saturate(1 - f^2)$ and modulate the specular and reflection contributions by k_a.

View-Dependent Streaky Reflections. When moving around any city streets late night during a rain, one of rain's strongest visual cues are the stretched reflections of various objects (such as street lamps, vehicle headlights, and so on) (Figures 1 and 13). These reflections are very prominent in any rainy scene. They appear to elongate toward the viewer, distorting the original reflecting object vertically proportional to the distance from the viewer. Water in the puddles and on the streets warp the reflections. This subtle effect greatly increases the realism of the rainy environment, especially as the falling raindrops hitting the puddles create dynamic warping of the reflections. The original shape of the reflector is distinguishable only by the blurred dominant colors of the reflecting object. Thus we want to preserve the

Figure 13: *View-dependent streaky reflections*

principle colors and create a blurry glowing reflection image for each reflecting object or light source in the scene.

We render reflections of complex arbitrary objects by using their impostors approximating the geometry of the reflected scene. We render the reflector objects into billboard reflector impostors (as described in [MS01]) both for the bright light sources and the dark objects (such as the telephone poles)). We render the impostors lit with the scene illumination using manually-simplified material shaders to ensure the accurate reflections appearance, however, the reflections materials shaders strongly saturate the dominant colors. The dynamic lighting allows us to represent reflected animated light sources (such as a flickering neon light or blinking traffic lights in the streets) correctly. The reflections attenuate simultaneously with their corresponding reflector objects.

The reflection impostors are dynamically stretched view-dependently toward the viewer in the vertex shader. The amount of stretching varies depending on the distance of the object to the viewer. The reflection buffer is down-scaled to half size of the original rendering buffer, and we use HDR texture formats to preserve the range for the reflections. The post-processing blurring technique (section 6.2) is used to dynamically streak the reflection buffer in the vertical direction to simulate warping due to raindrops striking in the puddles. Note that this is done in separate passes from the regular scene post-processing. The downsampling of the reflection buffer provides additional blurring for the reflections. To render objects with the stretched reflections, we sample from the reflection buffer using the screen space projection of the input vertex coordinate for each reflective object. We use object's per-pixel normal in tangent space to account for stretching of the reflection in view space and distort the reflection based on the surface normal. The post-process-based blurring and further warping alleviates specular aliasing and excessive flickering from reflections which would otherwise be highly distracting.

7. Results

Our test system consisted of rendering several city blocks in stormy weather, with animated vehicles and other objects in the scene. We used DirectX 9.0c© HLSL shaders to implement all of our effects, and the Lua scripting language to create the rendering scripts for the post-processing system and lightning integration. For rendering rain-related effects, nearly 300 unique shaders were used, with more than 500 used to render the entire complex environment in full. The rain-related shaders included various object shaders for wet materials, dynamic water simulation shaders, view-dependent reflections, raindrops, rain splashes, misty halos around objects, composite rainfall layer rendering, water droplet rendering and so on. Although our example video contains rendering of a night scene, the approaches presented in this paper can be successfully used in variety of lighting environments, including daytime renderings. The environment geometry, textures and rain-related offscreen buffers used 240 MB of video memory. In order to create a realistic environment, high resolution textures were used to capture the extreme detail of the represented world. For rendering individual falling raindrop and their splashes we used from 5,000 to 20,000 particles depending on a particular scene.

Effect	Frame rate	Rendering time
Composite rainfall	243 fps	4.11 ms
Raindrop particles (5-10K)	51.46 fps	19.45 ms
Raindrop splashes (5K)	52 fps	20.01 ms
Misty object halos	52.84 fps	18.93 ms
Raindrop splatters	285 fps	3.50ms
Post-processing for glow due to atmospheric scattering	414.03 fps	2.41 ms
GPU-based water simulation	360.18 fps	2.75 ms
GPU-based water rendering (includes simulation)	143.49 fps	6.98 ms
Droplet simulation	281.18 fps	3.55 ms
Rendering objects with droplets (includes simulation)	152.35 fps	6.59 ms
View-dependent reflections	114.48 fps	8.74 ms
Complete system rain rendering	32 fps	31.25 ms
Rendering without the rain effects	62.54 fps	15.71 ms

Figure 14: *Average performance results for rain rendering effects for full-screen scenes from the video rendered at 1024 x 768 resolution with 4X multisampling enabled*

Performance. We measured performance of our system on a 1GB Dual 3.2GHz Pentium 4 PC with a ATI Radeon X1900 XT graphics card with 512MB of video memory. Using our system on a complex environment described above, we achieve frame rates of 26-69 fps for the final rendering (shown in the accompanying video) depending on the com-

plexity of a specific scene and a combination of rain effects (see Figure 14 for the individual effect timings). Note that the rendering time for the raindrop particles and splash was limited by the CPU as the particle system simulation was CPU-bound.

8. Conclusions

Convincing and visually pleasing rendering of rain effects enhances the realism of outdoor scenes in many applications. In this paper we described a comprehensive system for interactive rendering of rain effects in real-time in complex environments. We presented a number of novel effects such as the image-space rainfall rendering, GPU-based water simulation for dynamic puddle rendering, water droplet animation and rendering using graphics hardware and view-dependent wet reflections, amongst all. All of the these effects help us generate an extensive, detail-rich urban environment in stormy weather. We hope that the new technology can be successfully used in the next generation of computer games and real-time rendering.

9. Acknowledgements

We would like to thank the ATI ToyShop team for their work on the interactive demo shown in the video. The programmers (also responsible for the rendering engine along with the authors): Daniel Ginsburg, Thorsten Scheuermann, Chris Oat, David Gosselin, the artists: Dan Roeger, Daniel Szecket (who have contributed many algorithm ideas for the puddle effects), Abe Wiley and Eli Turner; and the producers (Lisa Close and Callan McInally). We are also very grateful to Thorsten Scheuermann for the initial idea of the GPU water simulation and Chris Oat for his implementation of the Kawase post-processing method.

References

[BSS*04] BORSHUKOV G., SABOURIN K., SUZUKI M., LARSEN O., MIHASHI T., FAIMAN K., SCHINDERMAN S., JAMES O., JACK J.: Making of the superpunch. *ACM SIGGRAPH Sketches* (2004).

[DCWS03] DORETTO G., CHIUSO A., WU Y. N., SOATTO S.: Dynamic textures. *International Journal of Computer Vision 51*, 2 (2003), 91–109.

[dH57] DE HULST H. C. V.: *Light Scattering by Small Particles*, 1st ed. Dover, New York, NY, 1957.

[ENN90] EIHACHIRO NAKAMAE KAZUFUMI KANEDA T. O., NISHITA T.: A lighting model aiming at drive simulators. *ACM Trans. Graph. 24*, 4 (1990), 395–404.

[FHP99] FOURNIER P., HABIBI A., POULIN P.: Simulating the flow of liquid droplets. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 121–128.

[FM96] FOSTER N., METAXAS D.: Realistic animation of liquids. In *Graphics Models Image Process* (1996), vol. 58, pp. 471–483.

[GN04] GARG K., NAYAR S.: Detection and removal of rain from videos. *IEEE Conference on Computer Vision and Pattern Recognition* (2004), 528–535.

[GN06] GARG K., NAYAR S.: Photorealistic rendering of rain streaks. To appear in the proceedings of ACM SIGGRAPH 2006, August 2006.

[Gom00] GOMEZ M.: Interactive simulation of water surfaces. In *Game Programming Gems*, De Loura M., (Ed.). Charles River Media, Rockland, MA, 2000, ch. 2.6, pp. 185–194.

[Har03] HARRIS M. J.: *Real-time cloud simulation and rendering*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, 192003.

[Her01] HERMAN D. L.: Rainman: Fluid pseudodynamics with probabilistic control for stylized raindrops. *ACM SIGGRAPH Sketches* (2001).

[Jen01] JENSEN H. W.: *Realistic Image Synthesis Using Photon Mapping*, 1st ed. A K Peters, Ltd., Natick, MA, 2001.

[Kaw03] KAWASE M.: Frame buffer postprocessing effects in double-s.t.e.a.l (wreckless). GDC 2003 lecture, San Jose, CA, March 2003.

[KIY99] KANEDA K., IKEDA S., YAMASHITA H.: Animation of water droplets moving down a surface. *Journal of Visualization and Computer Animation 10*, 1 (1999), 15–26.

[LPFH01] LENGYEL J., PRAUN E., FINKELSTEIN A., HOPPE H.: Real-time fur over arbitrary surfaces. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (New York, NY, USA, 2001), ACM Press, pp. 227–232.

[Mas75] MASON B. J.: *Clouds, Rain and Rainmaking*. Cambridge Press, 1975.

[MS01] MACIEL P., SHIRLEY P.: Visual navigation of large environments using textured clusters. In *Symposium on Interactive 3D Graphics* (2001), pp. 95–102.

[NN03] NARASIMHAN S. G., NAYAR S. K.: Shedding light on the weather. In *IEEE CVPR* (2003).

[Sta99] STAM J.: Stable fluids. In *SIGGRAPH 99* (August 1999), ACM SIGGRAPH, pp. 121–128.

[SW03] STARIK S., WERMAN M.: Simulation of rain in videos. *International Journal of Computer Vision Texture 2003 (The 3rd international workshop on texture analysis and synthesis)* (2003), 95–100.

[WC75] WANG T., CLIFFORD R. S.: Use of rainfall-induced optical scintillations to measure path-averaged rain parameters. *JOSA* (1975), 8–927–237.

[WMT05] WANG H., MUCHA P. J., TURK G.: Water drops on surfaces. *ACM Trans. Graph. 24*, 3 (2005), 921–929.

[WW04] WANG N., WADE B.: Rendering falling rain and snow. *ACM SIGGRAPH Sketches* (2004).

[YYT01] Y.DOBASHI, YAMAMOTO T., T.NISHITA: Efficient rendering of lightning taking into account scattering effects due to clouds and atmospheric particles. In *Pacific Graphics* (2001), pp. 390–399.