
SIGGRAPH2006

Artist-Directable Real-Time Rain Rendering in City Environments

Natalya Tatarchuk

3D Application Research Group
ATI Research

What's in It for You?

- Share our lessons of developing an extensive environment for next generation
  - To help you in similar research and development goals
  - The new technology developed for this demo
- Learn what it means to render rain in city environments
  - Novel algorithms
  - Lighting and lightning
  - Raindrops and puddles
- Lots of eye candy!

Overview

- Rain rendering: introduction
- Related work
- Lightning system
- Rendering rain precipitation
- Water effects – puddles and droplets
- Post-processing for glow and mist
- Wet reflections
- Results discussion


SIGGRAPH2006
Introduction

The Challenge of Rendering Rain

Photorealistic rain greatly enhances the scenes of outdoor reality, with applications including computer games and motion pictures. Creating a faithful representation of rain in complex natural environments is a non-trivial problem. The challenge 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 modeled 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.

Creating an Illusion of Rain in Games

- Rain consists of numerous effects interacting together
- Previously games focused on rendering rain with one or two components
  - Often only using simple alpha-blended particle systems for rain which follow the camera – doesn't look very realistic
  - That type of rain does not respond to dynamic lighting
  - Requires too many particles for a feeling of strong rainfall
- Recently, some were able to incorporate our rain effects directly into their games
  - "Heavy rain" by Quantic Dream – post-processing rain effects and water droplets

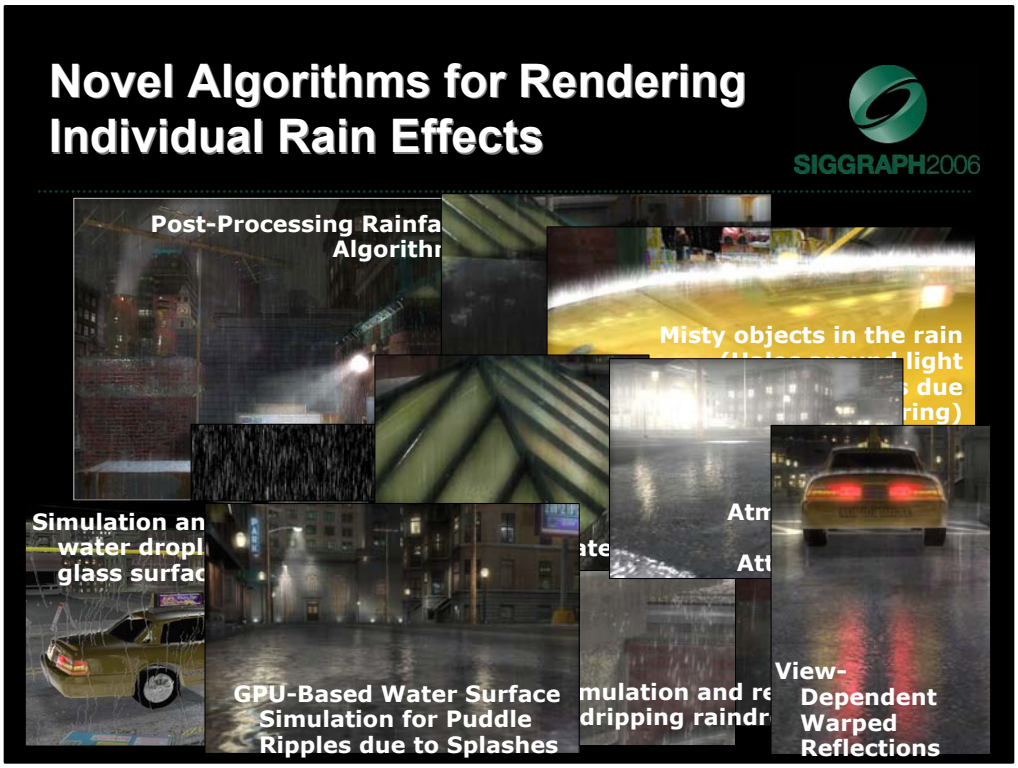
Rain is an extremely complex atmospheric natural phenomenon. It consists of numerous visual effects interacting together. Some recent games which incorporate rain rendering use simplistic approaches, including rendering stretched, blended particles to simulate falling raindrops or using blended animated 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. Simply rendering particle rain is not enough: Rain has a variety of visual cues, not just the streaky drops. Missing pieces can destroy the illusion of immersion

Contributions

- Need inexpensive and streamlined algorithms for photorealistic rendering of rainfall and rain-related effects
  - For games and interactive applications
- Our contribution is an intuitive, comprehensive and flexible system with numerous effects
  - Supports rendering rain effects in real-time in a complex environment
  - Provides a high degree of artistic control for the final look

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.



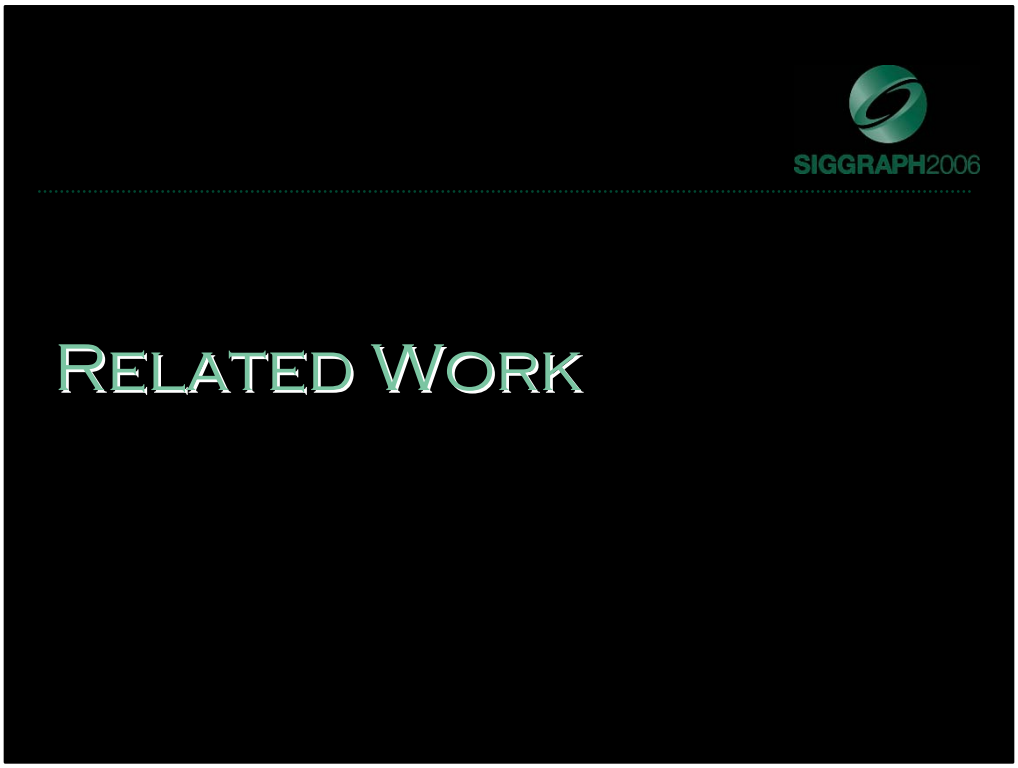
We present a number of novel algorithms for rendering the individual components of rain, including the following:

- 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

Rendering Complex Environment: Without Rain

Rendering Complex Environment: With Rain

Let’s take a look at the demo.


SIGGRAPH2006
Related Work

Related Work in Rain Precipitation

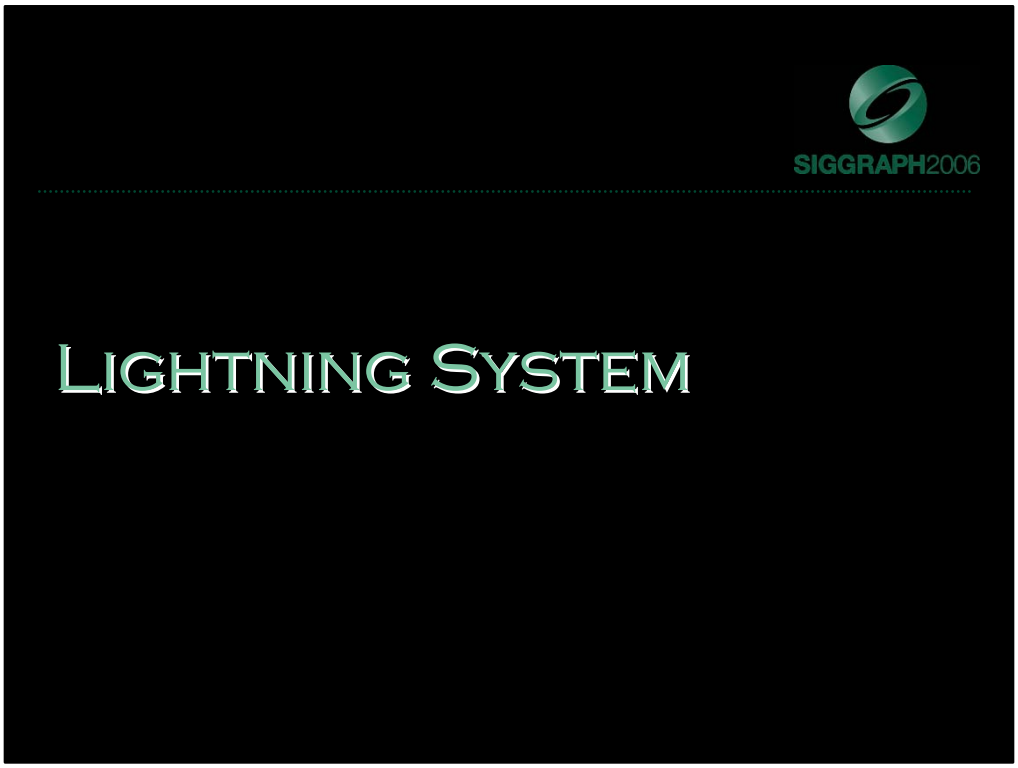
SIGGRAPH2006

- Much research on the rain phenomena in atmospheric sciences ([Mason75] and [Wang75])
- At the moment, only a few approaches exist for creating realistic rainfall rendering
  - Dynamically responding to lighting environment changes and camera movement
- Constant brightness rain strokes generated in [Starik03] for simulation of rain in videos
- Snow and rain precipitation in [Wang04]
  - Modeled with interpolated hand-drawn textures
  - With constant brightness
  - Mapped on a camera-aligned double-cone

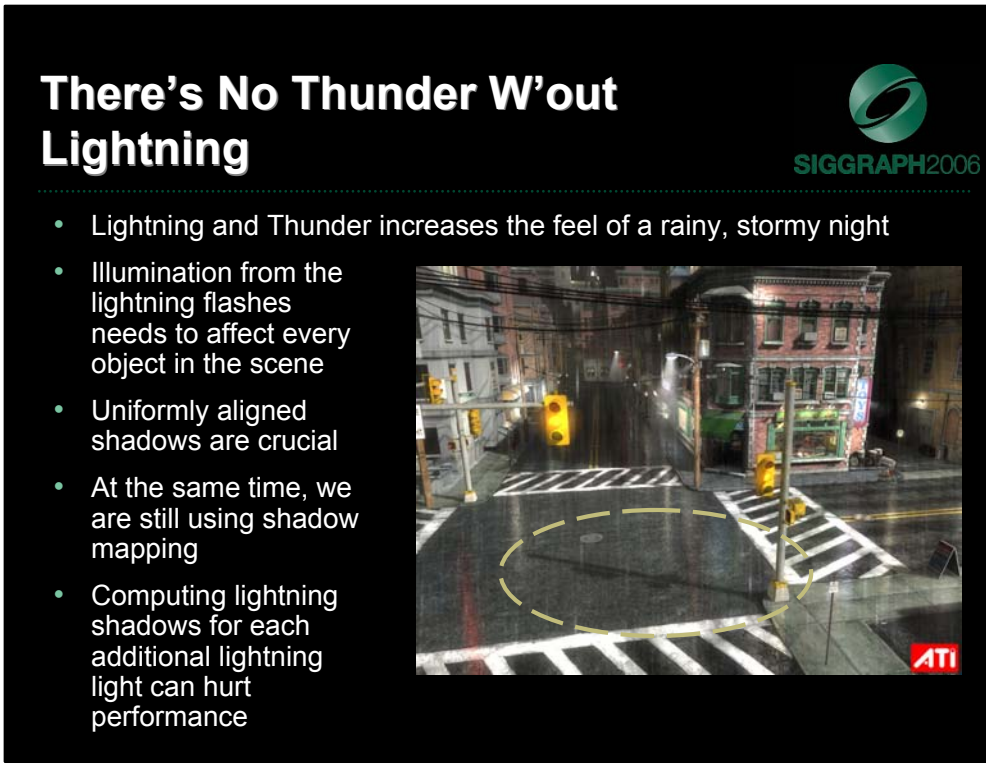
Rain effects have been examined in the context of atmospheric sciences, as well as in the field of computer vision. However, at the moment only a few approaches exist for creating realistic rainfall rendering that dynamically responds to the lighting environment and camera movement. Constant brightness 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 modeled 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.

Related Work Reference

- For a thorough review of related work in rendering rain components, look in
  - Tatarchuk, N., Isidoro, J. 2006. Artist-Directable Real-Time Rain Rendering in City Environments, in proceedings of *Eurographics Workshop on Natural Phenomena,* Vienna, Austria 2006


SIGGRAPH2006
Lightning System

There's No Thunder W'out Lightning



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. 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.

Lightning Challenges

- Lightning is a strong directional light that has to affect every object in the scene
- Lightning effect would feel very repetitive if it only comes from one direction
- The viewer can get very close to some of the lightning shadows…
  - Resolution has to hold up

Creating a convincing and realistic lightning effect is challenging for a variety of reasons:

•Lightning is a strong directional light that has to affect every object in the scene.

•Lightning effect would feel very repetitive if it only comes from one direction.

•The viewer can get very close to some of the lightning shadows… resolution has to hold up.

•Needs to work seamlessly with the other lighting solutions in our scene, such as our depth mapped shadow casting lights.

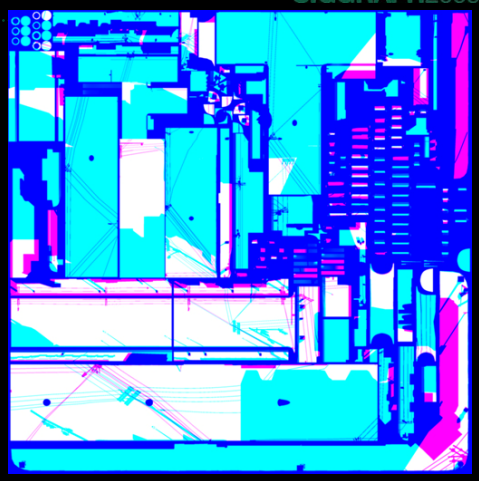
•Computing this type of lighting effect at run time would be a huge performance hit.

Lighting System Solution

- In-engine system driving lightning flashes during rendering
  - Via rendering script (in Lua)
- Support multiple simultaneous lighting flash light sources
  - Limiting factor – memory footprint

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.

Solution: Lightning Maps



- Implemented lightning as 2 unique lightning lightmaps
- Encode scene lightning information into maps during preprocessing
- Artist editable intensity for custom mixing of these two maps
- Pre-rendered in Maya and stored as a 8-bit grayscale image

During the preprocessing phase for the environment several key lightning source directions are picked by the artists and global 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.

Implemented lightning as 2 unique lightning lightmaps:

- One at an angle and one closer to directly above the scene (slightly offset)

- Artist editable intensity parameter to allow for custom mixing of these two maps.

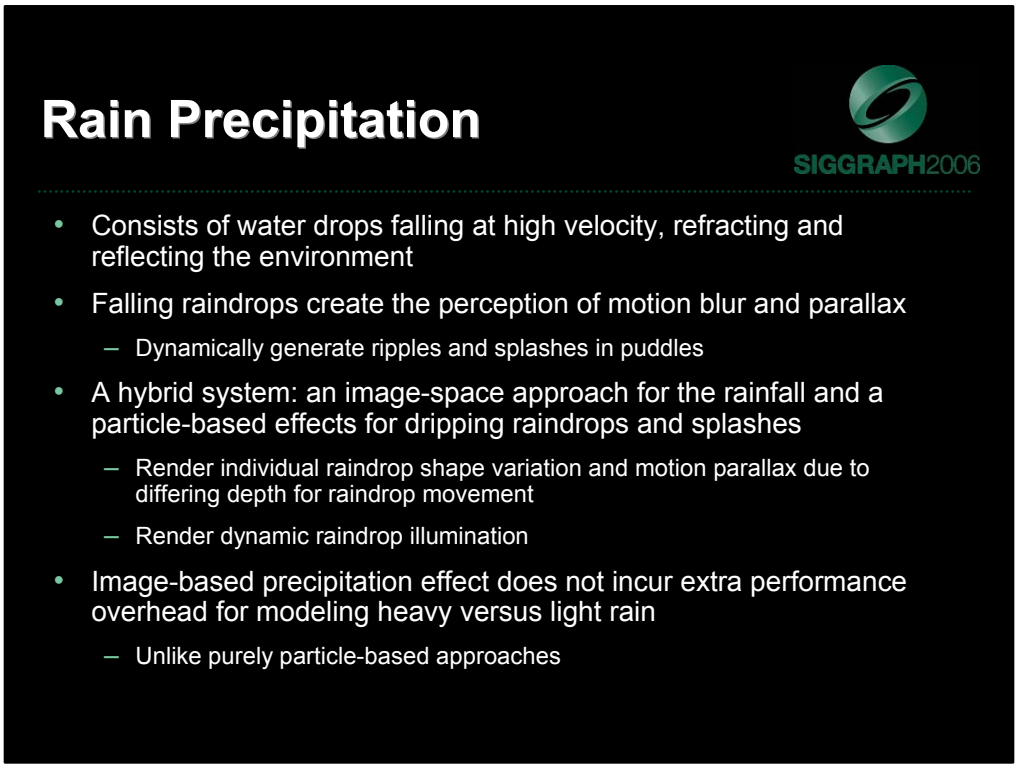
Mixing these two maps in different ways makes it seem like we have a wider variety of lightning direction than we actually do. Pre-rendered in Maya and stored as a 8bit grayscale image. Only needs to be 8bit as it is only used as an intensity value multiplier of the underlying HDR lightmap. Does not need any additional tone info. Packed into R and B of a single RGB image to reduce draw calls.

This value is also piped into other shaders, the rain for example. Used lightning intensity to adjust the opacity of the rain. Use either or both lightning brightness parameter or lightning lightmap sample added to the regular lightmap sample before tone mapping

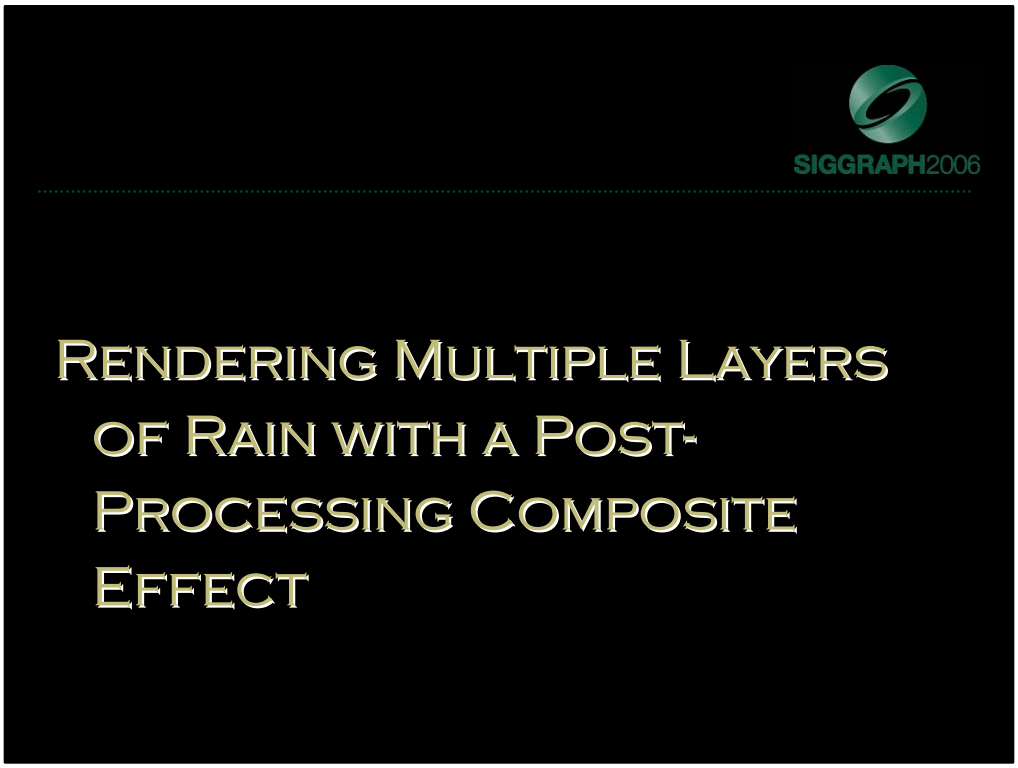
Incorporating Illumination from Lightning

- Every shaded pixel uses lightning illumination information
- The script propagates these to all object shaders
  - A lightning brightness parameter
  - Lightning illumination is added to the regular illumination for each material prior tone mapping
  - Negligible performance cost: cheap and effective
    - 1 additional texture fetch plus a couple of ALUs
- All object materials use this model and appear to respond accurately to lightning illumination
- Translucency of water is affected by lightning strength
  - Used the lightning brightness value to adjust the pixel's opacity as well for rain effects


SIGGRAPH2006
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 modeling heavy versus light precipitation.



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

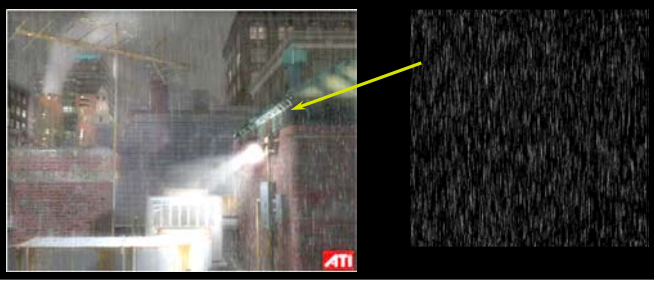
- Render composite rainfall layer prior to the final scene post-processing
  - Rendered as a full-screen quad over the scene
- Must consider performance implications
  - Every pixel on the screen goes through the rain shader
- Need minimize distracting repeating patters for rainfall
  - All the while keeping shader complexity minimal

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.

Raindrops behave like lenses refracting and reflecting (both specularly and internally) scene radiances towards the camera. Tangent space is specified by the view matrix (since it's a full-screen quad)

Creating Rainfall

- The artists specify the rain direction and speed in world-space to simulate different rainfall strength
  - Initial raindrop distribution is simulated with an animated 8 bit rainfall position placement texture
  - To simulate raindrop mistiness we blur the rain layer using the post-processing pipeline



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.

Moving Per-Frame Raindrop Distribution

- Rain direction is moved into clip space
- For each pixel we determine each raindrop's distribution position in clip-space as follows:
  - Using current time step Δt and the rainfall speed v_r

$$(x_i, y_i) = v_r^{cp} \cdot |v_r| \cdot \Delta t$$

- Sample from the raindrop distribution texture
- So far fairly straight-forward
  - But this alone would generate strong repeating patterns!
  - Undesirable

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 presented.

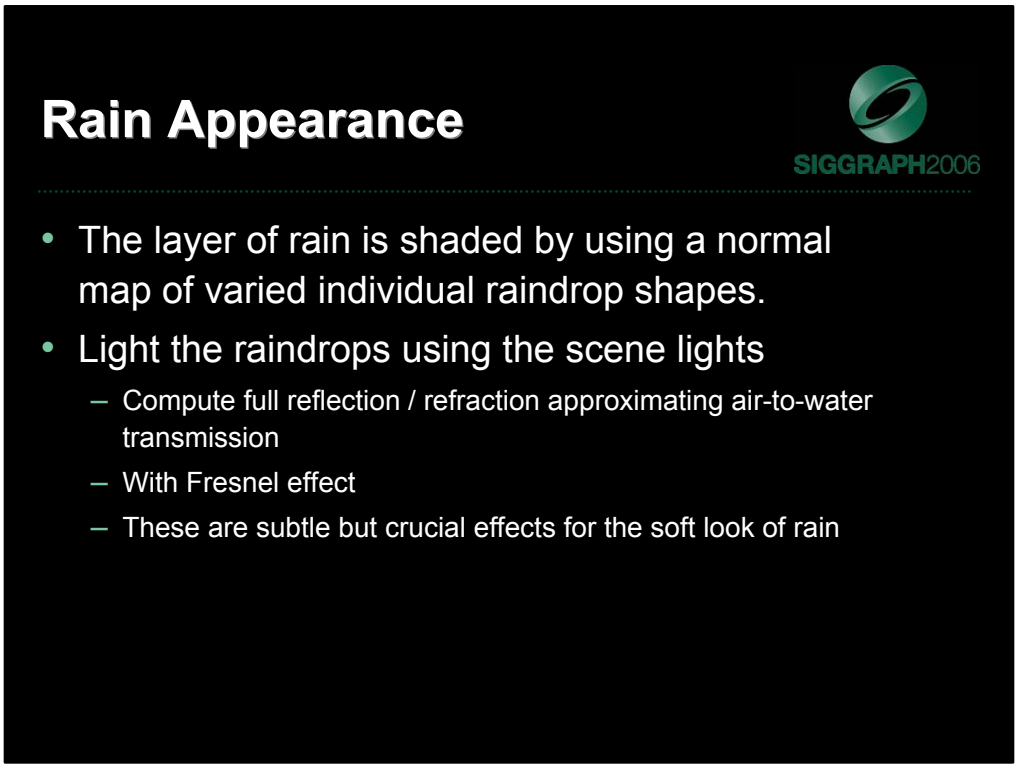
Simulating Multiple Layers of Rain

- Artists specify a rain parallax parameter p_r
  - Maps the depth range for rain layers
- Compute a randomized value for an individual raindrop r_i
  - Using the concepts of stochastic distribution for simulation of dynamic textures
- Model multiple layers of rain in a single pass with a single texture fetch from the rainfall position placement texture
  - Using p_r, r_i, and the screenspace raindrop location (x_i, y_i)
  - $p_r * r_i$ used as the w for a projective texture read
- Allows simulation of raindrops falling with different speed at different layers

We simulate multiple layers of rain moving with different speeds at varied depths rendered in a single geometric layer. 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.



We use a normal map for falling raindrops to model illumination for each raindrop in the composite layer. 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.

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 its background (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. Thus, in spite of being transparent, the average brightness within a stationary drop (*without motion-blur*) does not depend strongly on its background. Falling raindrops produce motion-blurred intensities due to the finite integration time of a camera. These intensities are seen as *streaks* of rain. Unlike a stationary drop, the intensities of a rain streak depend on the brightness of the (stationary) drop as well as the background scene radiances and integration time of the camera.

Rain Appearance

- The layer of rain is shaded by using a normal map of varied individual raindrop shapes.
- Light the raindrops using the scene lights
- The rain must correctly respond to lightning illumination
  - Use lightning brightness parameters to bias reflection and refraction for water effects
  - Lightning brightness also adjusts the opacity and the glow amount
- Our approach does not require any preprocessing and can handle an arbitrary number of light sources

We use a normal map for falling raindrops to model illumination for each raindrop in the composite layer. 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.

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 its background (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. Thus, in spite of being transparent, the average brightness within a stationary drop (*without motion-blur*) does not depend strongly on its background. Falling raindrops produce motion-blurred intensities due to the finite integration time of a camera. These intensities are seen as *streaks* of rain. Unlike a stationary drop, the intensities of a rain streak depend on the brightness of the (stationary) drop as well as the background scene radiances and integration time of the camera.

Creating the Feeling of Strong Rain

- Realistic rain is very faint in bright regions of the scene and tends to appear stronger when light falls in a dark area
  - If this is modeled exactly, the rain appears too faint
- Simulate an old Hollywood trick for rain on film instead
  - The film crew add milk to water to make rain appear stronger on film
  - We do the same, by biasing rain color and opacity to appear whiter
  - Although exaggerated, this creates a *perception* of stronger rainfall

Realistic rain is very faint in bright regions but tends to appear stronger when the light falls in a dark area. Physically accurate modeling 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.


SIGGRAPH2006
Rendering Dripping Raindrops

Raindrop Particles Rain

- To simulate raindrops falling off various objects in our scene, we used billboard particle systems
- Artist-placed and animated based on a ‘template’ system
- To render each *individual* particle:
  - Stretch billboard based on velocity
  - Use a normal map for a droplet (a blurry version of a full raindrop’s normal map)
  - Tangent space is defined by the view matrix
  - Only compute reflection and refraction to simulate air-to-water transmission
  - Droplets should appear more reflective and refractive when the lightning strikes
  - Biased lightning brightness value adjusts the refraction / reflection color

During rain, raindrops drizzle from various objects in the scene - trickling off gutter pipes, window ledges and so on.

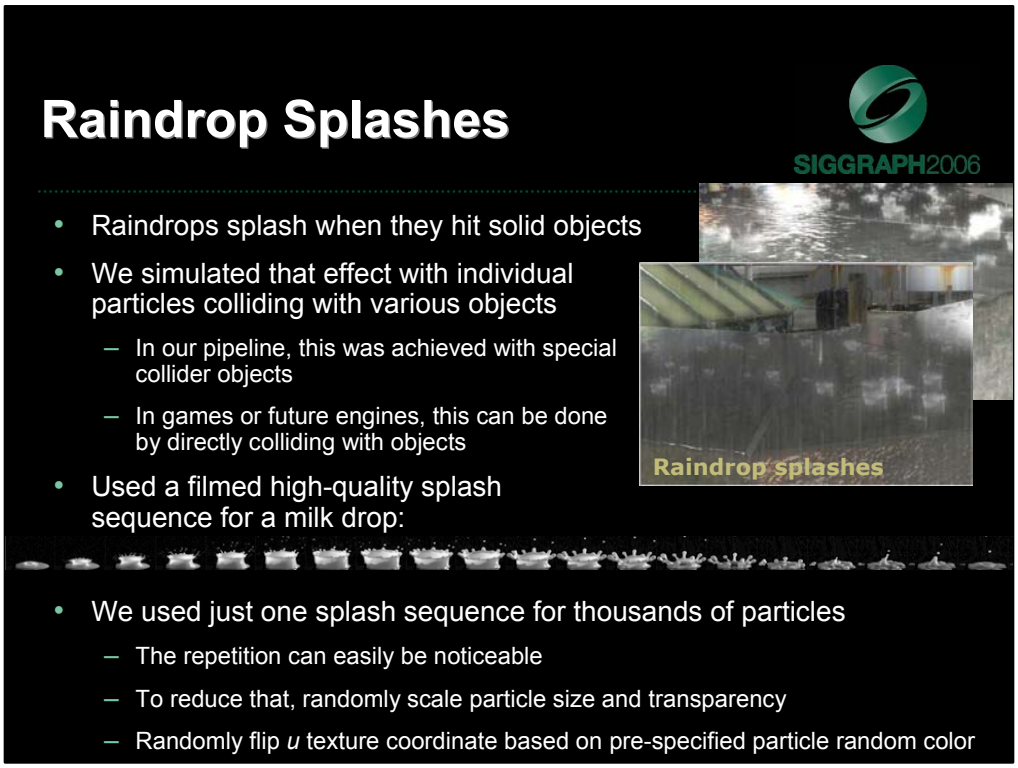
We simulate this effect with the use of physics-based particle systems using screen-aligned billboard representation for individual raindrops. The base particle system simulation uses the physical forces of gravity, wind and several animation parameters for raindrop movement. The artists can place any number of separate particle systems, culled by the camera frustum during rendering, throughout the environment to generate dripping raindrops.

Controlling Raindrop Transparency

- We attenuate raindrop opacity by distance
- Attenuate the opacity by Fresnel scaled and biased by artist-specified edge strength and bias
  - To make the raindrop appear less solid and billboard-like
- Observation: Raindrops should appear more transparent (like water) when the lightning strikes
  - Scaling the opacity by 1 – ½ * lightning Brightness does the trick
  - The particles still appear their respective transparency when there is no lightning
  - They become more translucent-like when the lightning strikes
  - This was used for both raindrop particles and raindrop splashes to attenuate their transparency


SIGGRAPH2006
Raindrop Splashes

Raindrop Splashes



- Raindrops splash when they hit solid objects
- We simulated that effect with individual particles colliding with various objects
  - In our pipeline, this was achieved with special collider objects
  - In games or future engines, this can be done by directly colliding with objects
- Used a filmed high-quality splash sequence for a milk drop:
- We used just one splash sequence for thousands of particles
  - The repetition can easily be noticeable
  - To reduce that, randomly scale particle size and transparency
  - Randomly flip *u* texture coordinate based on pre-specified particle random color

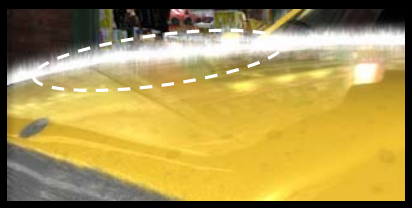
We utilize the particle systems for rain drop splashes off the surface of objects, using pre-rendered high-quality splash sequence for a milk drop.

Illuminating the Splashes

- Splashes should appear correctly lit by the environment lights
  - If light sources were behind the rain splashes, rendered the splashes as brightened backlit objects
  - Otherwise just used simple ambient lighting
  - This worked particularly well when rendering under bright sources
  - Compute specular lighting in the vertex shader for all lights
- We used an overhead lightmap to simulate sky and street lamp lighting
  - Use the splash world-space position as coordinates to look up into this lightmap (with some scale and bias)
  - The overhead lightmap value modulated splash illumination

Used lightning brightness to both adjust the lighting for the splash and its opacity (as described before)

Misty Object Halos Due to Rain Precipitation



- In a strong rainfall, raindrops generate delicate halo outlines around the edges of objects
- This effect can be generated by using the “fins” technique borrowed from the real-time fur rendering [Lengyel01]
- 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 quad by using an algorithm similar to the rainfall algorithm

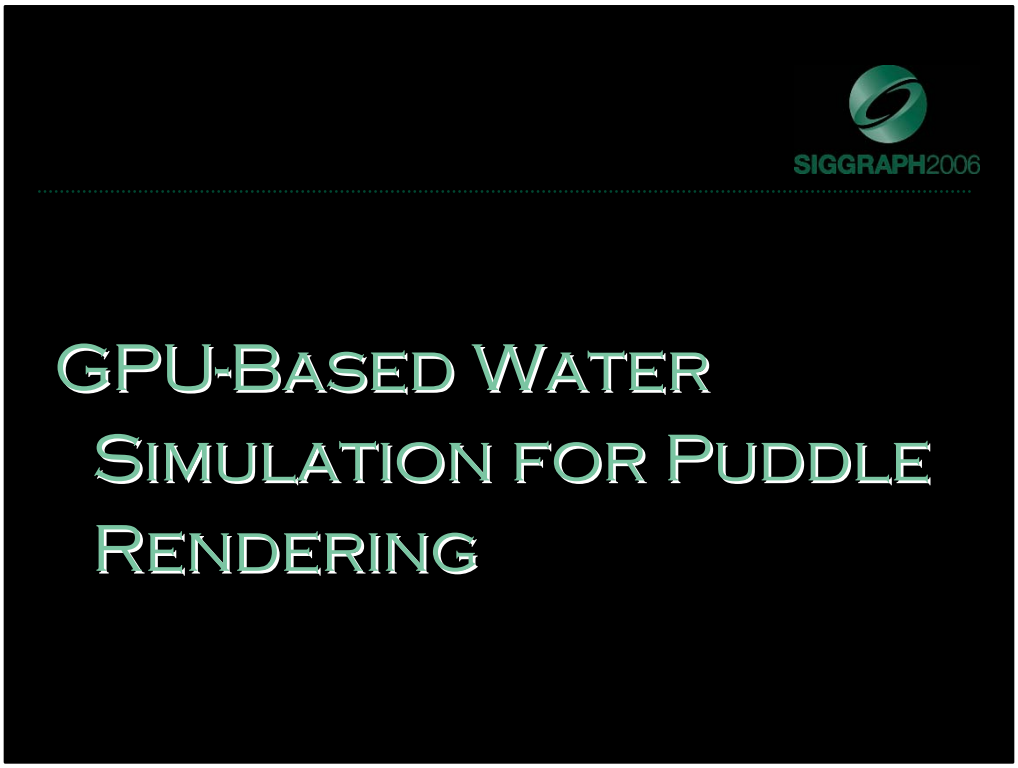
In a strong rainfall, as the raindrops strike solid objects, they generate not only the splashes, but also the delicate halo outlines along the edges of objects. This is a very important visual cue which has been omitted from most of the existing rendered rain environments. We support rendering of this effect for objects in our scene (including the animated objects, such as cars 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 as an animated texture, alpha-blended with the rest of the environment.

Rain Splatters

- Strong rainfall also generates an effect of raindrop splattering on the surface of wet materials
- We use a shells-based technique to create this effect
  - Again borrowing from the real-time fur rendering approach [Lengyel01]
- Rain splatters are rendered on the surface of objects in the form of concentric circles
  - In each successive shell expand the splash circle footprint with a series of animated texture fetches
  - Blend onto the previous shells

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. 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.


SIGGRAPH2006
Demo: The Taxi


SIGGRAPH2006
GPU-Based Water Simulation for Puddle Rendering

Water Ripples in Puddles

- Goal: Dynamic *realistic wave motion of interacting ripples* over the water surface
  - With fast simulation directly on the GPU
- Water ripples are generated as a result of rain drops falling onto the geometry in the scene
  - We support generation of raindrop ripples as a result of direct collision
  - However, for our scene that would require too much memory
- Instead, we use a stochastic seeding method for the simulation
  - Seeding rain drops into a texture
  - Spatter raindrops as points into the water simulation texture
- Can also render object outlines into the seeding texture to generate wakes

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. 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.

Pass 1: Render seeds into the first water simulation buffer

These seeds rendered as initial positions of water ripples

The seeds 'excite' ripple propagation

Pass 2 and Pass 3: Perform integration on water surface simulation

Uses 'ping-pong' texture feedback approach

We only use two passes, but more will help with system stability if time step desired to be smaller

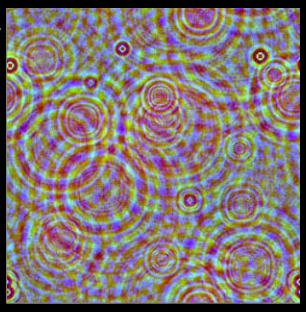
These passes generate water height field

Pass 4: Generate water normals

Sample from the water normals texture when rendering an object with puddles

Water Surface Approximation

- Approximate water surface with a lattice of points
  - We render our surfaces with a 256 x 256 simulation
  - Each lattice contains information about water
    - Current position as a height value
    - Previous time step's position
- We simulate the water lattice entirely on GPU
  - Using a texture to store lattice positions and its attributes
  - Similar to "Interactive Simulation of Water Surfaces" by M. Gomez (*Game Programming Gems*)
  - However, there the lattice is approximated with vertices on the CPU



Water lattice heights: Current frame's height in R channel and previous frame's height in G channel

We approximate the water surface as a lattice of points on 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. This would generate a wake effect in the water surface.

Simulate Water Interaction

- Real-life raindrops generate multiple ripples that interact with other ripples on the water surface
  - We implement the same model
- Single rain drop generates ripples with damping for the duration of the splash life span
- Render a raindrop into the wave height texture using a dampened sine wave
  - As a function of the raindrop mass
  - Approximates the concentric circular ripples

Similar to real-life raindrops, a single raindrop in our system excites multiple interacting ripples on 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 a water puddle.

Water Surface Response

- Treat water surface as a thin elastic membrane
  - Ignore gravity and other forces
  - Only account for surface tension
- At every time step, infinitesimal sections of this surface are displaced
  - Due to tension exerted from their direct neighbors
  - Acting as spring forces to minimize space between them

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.

Computing Ripple Heights

- Vertical height of each water surface point can be computed with partial differential equation:

$$\frac{\partial^2 z}{\partial t^2} = v^2\left(\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2}\right)$$

  where *v* is the velocity of the waves traveling across the surface

- Solve this equation in real-time to determine water wave height for each point in the lattice
  - PDE solution is computed with explicit Euler integration in SM 2.0 pixel shaders
  - Two passes gave us sufficiently stable simulation
- During the final pass, compute the water normals using a Sobel filter on the wave heights

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.

Integrating Water Puddles

- We render a single water simulation for the entire demo
  - All objects with water puddles sample from that (for example, streets, rooftop ledge, etc)
- Sample from water membrane simulation using current position
  - Use the (xz) world-space coordinates for look-up
  - Scaled by the artist parameter to vary by size of membrane simulation (and size of ripples)
  - To reduce visual repetition, we rotate these coordinates by a pre-specified angle (ex: 15°)
  - Angle is specified per-object
- No additional geometry is required for water puddles

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 so for ripple waves and a rotational angle qo for the ripples look-up. The ripple simulation sample coordinates are rotated in texture space based on the specified object angle qo. 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. We also specify the puddle strength parameter to specify how much the ripple normals perturb the original bump map. This allows create different water motion for various objects

Puddle Placement and Depth

- To render deep puddles, we use just the water puddle normal sampled as just describe, along with color / albedo attributes of the object
- However, in real environment, puddle depth varies greatly
- To simulate that, we allow a puddle depth and location mask map
- Adding puddles with ripples to objects:
  - Define scale parameter and sample ripple normals
  - Sample puddle depth map
  - Interpolate between the normal map for the object and the water surface normal based on the puddle depth value

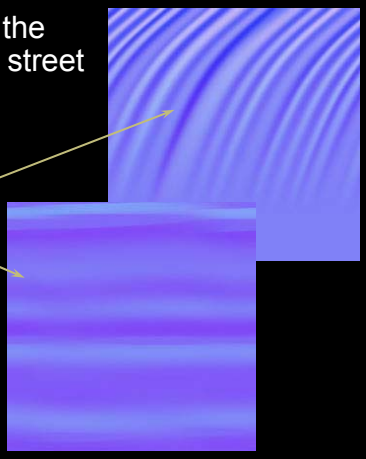
In real environments, water puddle depth and locations differ significantly due to landscape details and rainfall accumulation.

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 pi.

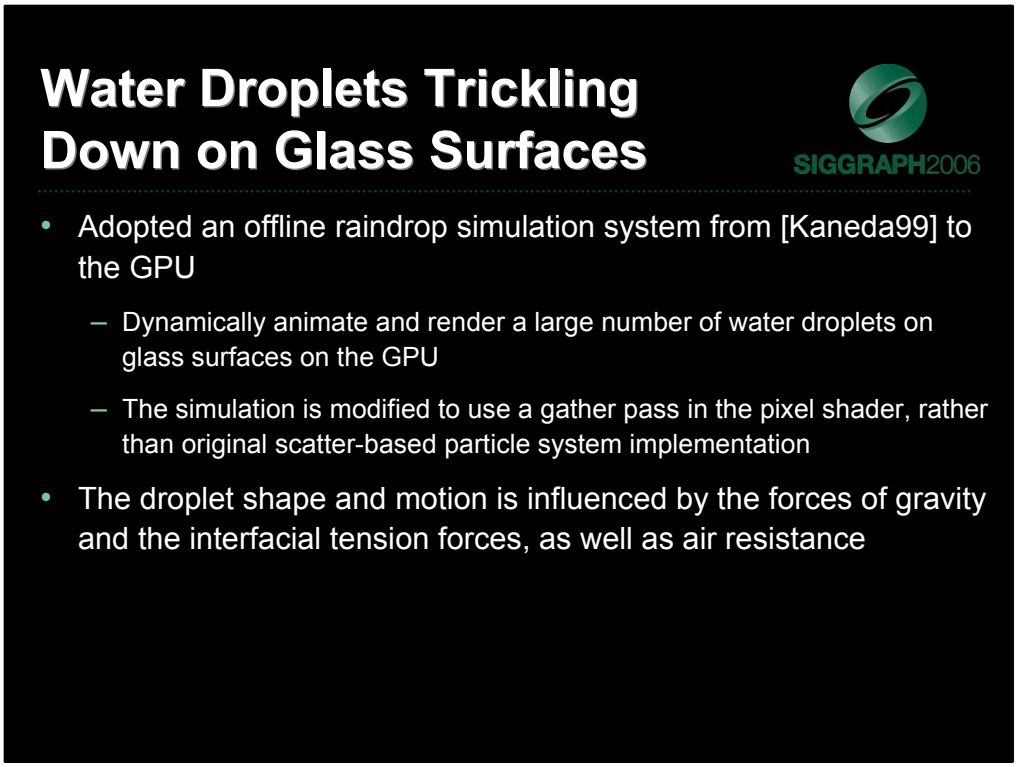
We interpolate between the object normal vector and the water surface normal based on di and an artist-specified puddle influence parameter p_i. Using this perturbed normal, we render the objects with water surfaces using Fresnel equations ([Jen01]) for water-air refraction and reflection, as well as the material properties of the object as desired. We also specify the puddle strength parameter to specify how much the ripple normals perturb the original bump map. This allows create different water motion for various objects

Creating Swirling Water Puddle

- Create an impression of water, swirling towards the drain, with ripples from the raindrops
- To create ripples from raindrops, use the same approach as for puddles on the street
- For water, swirling toward the drain, we used several ‘wake’ normal maps
  - Swirling radially around the drain
  - Concentric circles draining toward the drain



Water Droplet Animation and Rendering on Glass Surfaces in Real-Time on the GPU



We adopted an offline raindrop simulation system from Kaneda99 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. 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 traveling on the surface. Our system produces correctly lit droplet appearance including the refraction and reflection effects.

We run the more extensive droplet simulation on the main store front windows, whereas for other windows in the environment we have a version that only renders static droplets with some texture tricks to make them appear moving.

Droplet Movement

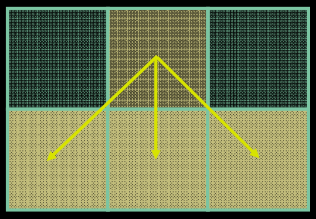
- The surface of the glass is represented by a lattice of cells

- Each cell contains:
  - The mass of water in each cell
  - Velocity in x and y
  - The droplet traversal quantity within the cell
  - Water affinity parameter for the surface
- The droplet information is packed into a 16-bit per channel RGBα texture

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). 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.

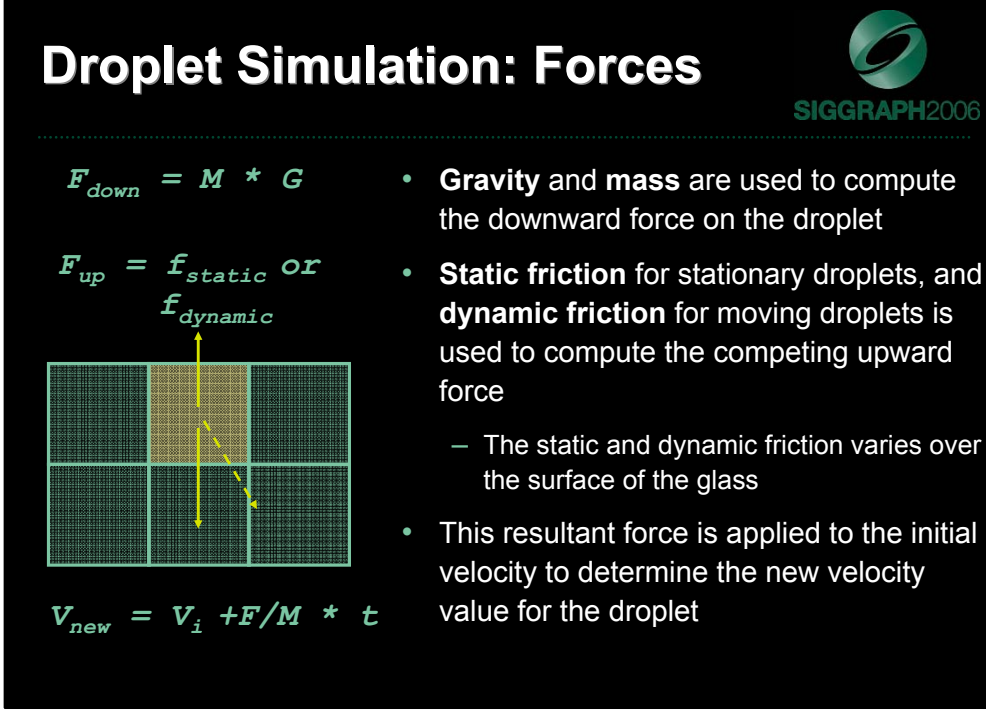
The droplet information is packed into a 16-bit per channel RGBa texture.

Droplet Movement Simulation



- Droplets can flow into one of the 3 cells below
- New cell to flow into is randomly chosen
  - Biased by velocity, friction based affinity and wetness of the target cell
- Droplets have a greater affinity for wet regions of the surface
- Velocity is updated based on target cell chosen

Droplet Simulation: Forces



- **Gravity** and **mass** are used to compute the downward force on the droplet
- **Static friction** for stationary droplets, and **dynamic friction** for moving droplets is used to compute the competing upward force
  - The static and dynamic friction varies over the surface of the glass
- This resultant force is applied to the initial velocity to determine the new velocity value for the droplet

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.

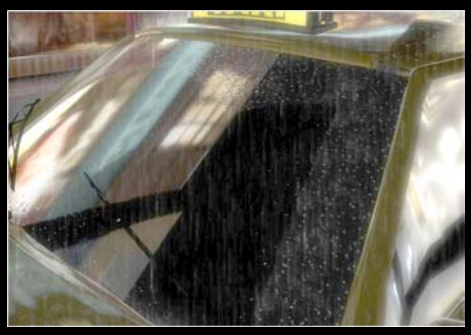
Droplet Rendering

SIGGRAPH2006

- After simulation, each cell contains a new mass value
- A bump map is derived based on this mass
  - This is used to perturb reflection & refraction vectors
- The droplet mass is also used to render dynamic shadows of the simulation onto the objects in the toy store
  - If the droplet mass is large enough, we render a ‘caustic’ highlight in the middle of the shadow for that droplet

The scene is rendered first and then we render the water droplet simulation on the window after that. This allows us to reflect and refract the scene through the individual water droplets. In order to do that, we use the water density for a given rendered pixels. If there is no water, we simply render the scene with regular properties. However, if the water is present, then we can use the water mass as an offset to refract through that water droplet.

Rain Effects - Windshield Wipers



- Wiper shader with droplets on glass of car
  - Wiper parameter is passed into the shader
  - Wiper maps are used to determine what regions have been recently swept clean
  - Two separate wiper maps so wiped regions can overlap


Demo: Water Droplets


SIGGRAPH2006
View-Dependent Streaky Reflections

We render bright objects into the half-size reflection buffer [1/4 for the roof]. We render a variety of objects, for example, all of the scene lamp, telephone pole lights, neon sign, traffic lights, etc. We specify the reflection buffer to be 1010102 as well so that we can preserve most of the light dynamic range (brighter than 1). In our case it's more 0-6.5 range. To save on draw-calls – all street lamps are rendered as one big object (using skinning) Use the same technique as we did for blurring the blow buffer to streak the reflection buffer and simulate water mistiness. We sample from the reflection buffer using screen space projection of the input vertex coordinate. We also use the normal in tangent space to account for stretching of the reflection in view space and warp the reflection based on the surface normal.

Rendering View-Dependent Reflections

- Render reflections of complex arbitrary objects by rendering their impostors
  - Approximating the geometry of the reflected scene
  - Render reflector impostors into a separate reflection buffer
- Render the reflector objects into billboard reflector impostors
  - For bright light sources and objects and for dark objects
  - Render the impostors fully lit with the simplified material shaders for accurate dynamic reflection appearance
  - Reflection material shader saturate the dominant colors

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.

Reflection Impostor Rendering

- Impostor geometry is dynamically stretched view-dependently toward the viewer in the vertex shader
  - The amount of stretching varies depending on reflector object's distance to the viewer
- Reflection buffer is downscaled to half size of the frame buffer
- Use HDR texture formats to preserve the dynamic range of reflections
- All reflector impostors are occluded as necessary by simple invisible blocker geometry

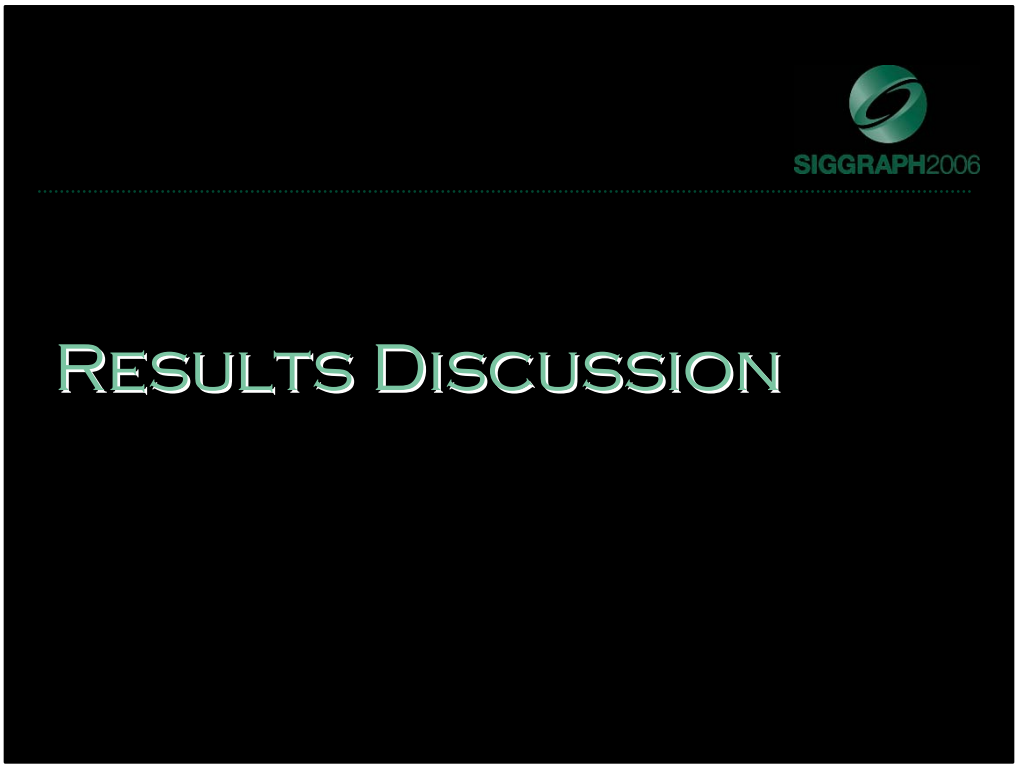
The reflection impostors are dynamically stretched view-dependently toward the viewer in the vertex shader. The amount of stretching varies depending on the distance of the object to the viewer. The reflection buffer is down-scaled to half size of the original rendering buffer, and we use HDR texture formats to preserve the range for the reflections. The post-processing blurring technique 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.

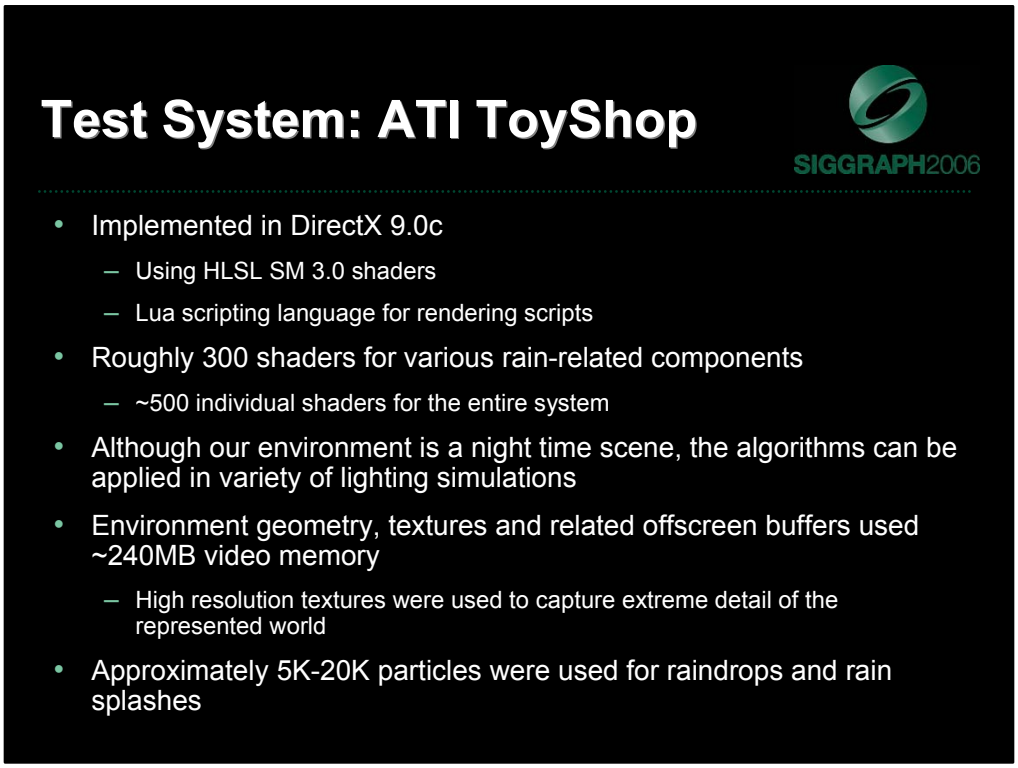
Incorporating Streaky Reflections

- During the final rendering of the scene, sample from the reflection buffer using the screen-space projection of the vertex
  - Use the object per-pixel normal in tangent space to account for stretching of the reflection in view-space
  - Warp the reflection based on the surface normal for the reflecting object
- To make the wet reflections appear blurry and streaky, we use the post-processing pipeline and apply a Kawase-style blurring in vertical direction only
  - This creates a feeling of misty reflections that are dynamically distorted by the raindrops
  - This is a separate pass from the regular scene post-processing
  - Additional benefit: this alleviates specular aliasing and excessive flickering of reflections

The post-processing blurring technique 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.


SIGGRAPH2006
Results Discussion



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 c 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.

Performance

- Measured on 1GB Dual 3.2 GHz Pentium 4 PC with ATI Radeon X1900 XT with 512MB of video memory
- Achieve frame rates of 26-69 fps
  - Rendering time for the raindrop particles and splashes was limited by the CPU

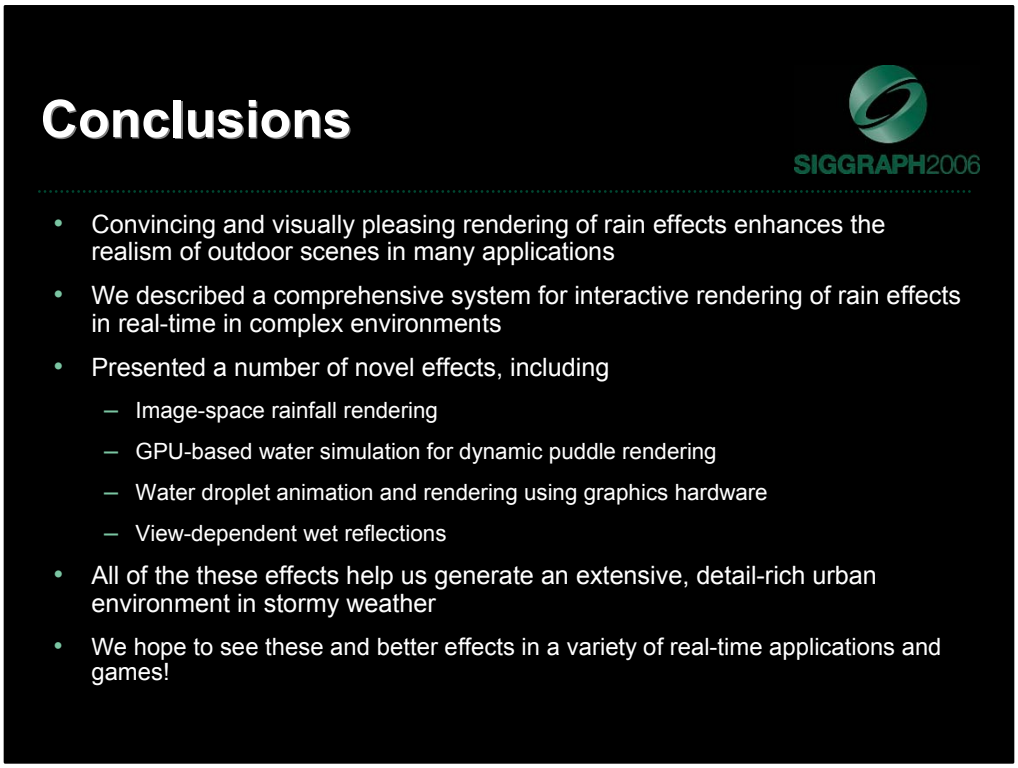
We measured performance of our system on a 1GB Dual 3.2GHz Pentium 4 PC with a ATI Radeon X1900 XT

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 complexity of a specific scene and a combination of rain effects (see next slide 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 running on the CPU and was CPU-bound.

Rain Components Rendering Times

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	3.50 ms
Post-processing for glow	4114.03 fps	2.41 ms
GPU-based water rendering (with simulation)	143.49 fps	6.98 ms
Rendering objects with droplets (includes simulation)	152.35 fps	6.59 fps
View-dependent reflections	114.48 fps	8.74 ms
Complete system rain rendering	**32 fps**	**31.25 fps**
Rendering w/out rain effects	**62.54 fps**	**15.71 ms**


SIGGRAPH2006
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.

Acknowledgments

- John Isidoro – for developing many of the rain effects described here and his great work on the ToyShop demo
- Thorsten Scheuermann for the initial GPU water idea
- David Gosselin for the idea of rendering the misty halo outlines using fur fins

Truly a team effort of which we are all very proud of.

Reference Material

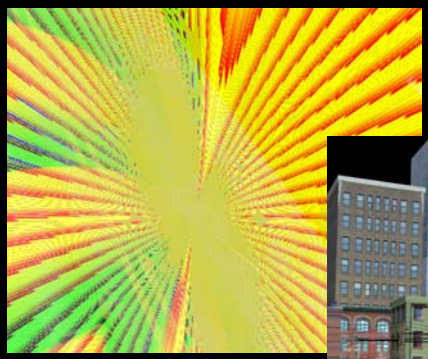
- www.ati.com/developer
  - All of these presentation and related materials
- http://www.ati.com/developer/techreports.html
  - Downloadable publications and videos from ATI Research
  - Tatarchuk, N. 2006. Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows, *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*
  - Tatarchuk, N., Isidoro, J. 2006. Artist-Directable Real-Time Rain Rendering in City Environments. Eurographics Workshop on Natural Phenomena, Vienna, Austria, September 2006.

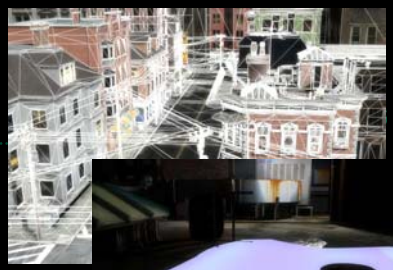
References

- [Garg04] GARG K., NAYAR S.: Detection and removal of rain from videos. IEEE Conference on Computer Vision and Pattern Recognition (2004), 528–535.
- [Garg06] GARG K., NAYAR S.: Photorealistic rendering of rain streaks. ACM SIGGRAPH 2006, August 2006.
- [Mason75] MASON B. J.: Clouds, Rain and Rainmaking. Cambridge Press, 1975.
- [Narasimhan03] NARASIMHAN S. G., NAYAR S. K.: Shedding light on the weather. In IEEE CVPR (2003).
- [Starik03] 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.
- [WW04] WANG N., WADE B.: Rendering falling rain and snow. ACM SIGGRAPH Sketches (2004).
- [Wang75] WANG T., CLIFFORD R. S.: Use of rainfall-induced optical scintillations to measure path- averaged rain parameters. JOSA (1975), 8–927–237.

Questions?

natasha@ati.com






Thank you!