






# Attention to Detail!

Creating Next Generation Content  
For Radeon® X1800 and beyond



**Callan McNally**  
Manager, 3D Application  
Research Group

# Overview

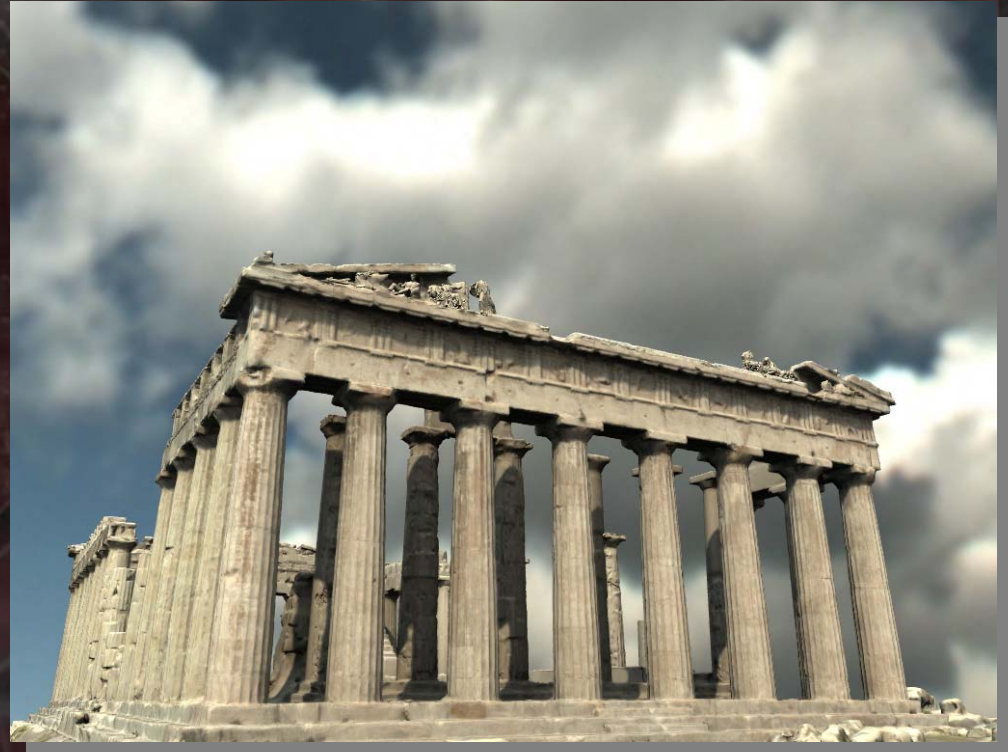


- In order to fully take advantage of next generation hardware, both artists and programmers will be required to pay more attention to the details.
- The cost of creating content is growing exponentially
  - Techniques are required to control the costs
  - While still allowing for increase in visual fidelity
- In our Parthenon and Toy Shop demos we take two different approaches to addressing this problem
- I'll highlight a few of the techniques we developed
- And also hopefully provide some insight into the challenges faced by developers

# Parthenon



- Based on Paul Debevec's work, first shown at Siggraph 2004
- The geometry for this demo comes from laser scans of the Parthenon at very high resolution.
- The source dataset is over 90 million polygons !!
  - That's a lot of detail to store, manage and render





# Parthenon Demo

# Managing 90 million polygons



- Data Set is first reduced to 15 million polygons
  - This fits with our performance goals
  - And has no visible difference at target resolutions
- A custom preprocess stage is performed
  - To re-parameterize the data into a form suitable for real-time rendering
  - To subdivide the model into individual clusters
- The model is simplified into multiple Levels of Detail
  - These LOD's are represented by a custom data format
  - Called "The Progressive Buffer"

# The Progressive Buffer



- Each Level of Detail in the model is represented by 2 buffers:
  - **Fine vertex buffer**  
Representing the vertices in the current LOD
  - **Coarse vertex buffer**  
Representing the vertices in the next lowest LOD.
    - Direct mapping exists between each vertex and the corresponding parent vertex
    - Requires Vertex Duplication

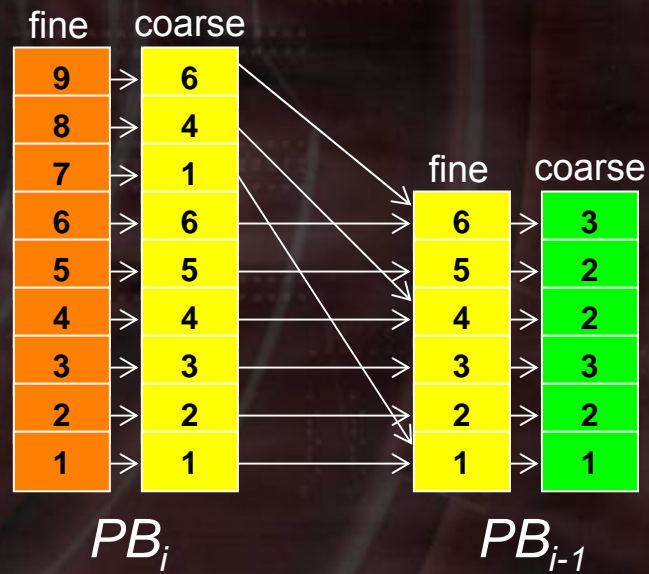


$PB_i$

# The Progressive Buffer



- At Runtime:
  - LOD levels to use are determined based on geometry distance to camera.
  - Both the fine and coarse buffers are streamed to the vertex shader.
  - Vertex shader smoothly blends position, normal and UVs. (blending weight based on vertex distance to camera)
  - Once all vertices in cluster are rendered using coarse buffer, switch to next lowest LOD representation.



The image features a central window with a white border and four corner handles. The window's background is dark and textured, with faint, glowing circular patterns. The text 'Progressive Buffers Demo' is centered in white. The window is set against a vibrant red background with a complex, futuristic design of concentric circles and glowing lines.

# Progressive Buffers Demo

# Parthenon

Additional details we don't have time for ...



- A similar technique to the Progressive Buffer is also used for Texture LOD management.
  - Could also be applied to Normal Maps but not required in this demo
- Only buffers currently in use are stored in VRAM
  - Others can be in main memory or even on disk.
  - Look ahead algorithm predicts when they will be needed and pre-loads on demand.
- Animated Skybox is an MPEG4 movie but retains HDR information for lighting and rendering.
- Custom shadow algorithm developed that uses temporal coherence to minimize cost.
- (TBD - 16 bit HDR rendering with AA - only if we use it)

# Toy Shop



- This demo is ALL detail and makes great use of the Radeon® X1800 hardware.
  - Rain + Water effects
  - Bumpy brickwork and cobblestones
    - using parallax occlusion mapping (POM)
  - Realistic Lighting
    - Volumetric Lights with Rain & Fog
    - Lightning - using 3DC+
    - Misty Rain Halos
    - Glows
  - Dynamic Stretchy, Blurry Reflections
  - Dynamic Soft Shadows
- Whole Scene approach to detail - not just for flypath
- Over 700 Unique Shaders used in this demo

# Dynamic Parallax Occlusion Mapping



- We want to render very detailed surfaces
- Don't want to pay the price of millions of triangles
  - Vertex transform cost
  - Memory footprint
- Want to render those detailed surfaces accurately
  - Preserve depth at all angles
  - Dynamic lighting
  - Self occlusion resulting in correct shadowing

# Previous Techniques



- Some solutions already exist to these problems
  - Bump Mapping
  - Per-pixel Normal Mapping
- However these have a number of limitations
  - Surfaces don't exhibit correct parallax behaviour
  - Self Shadowing is not supported
  - Sharp details aren't handled well
  - Illusion breaks down at Silhouette edges

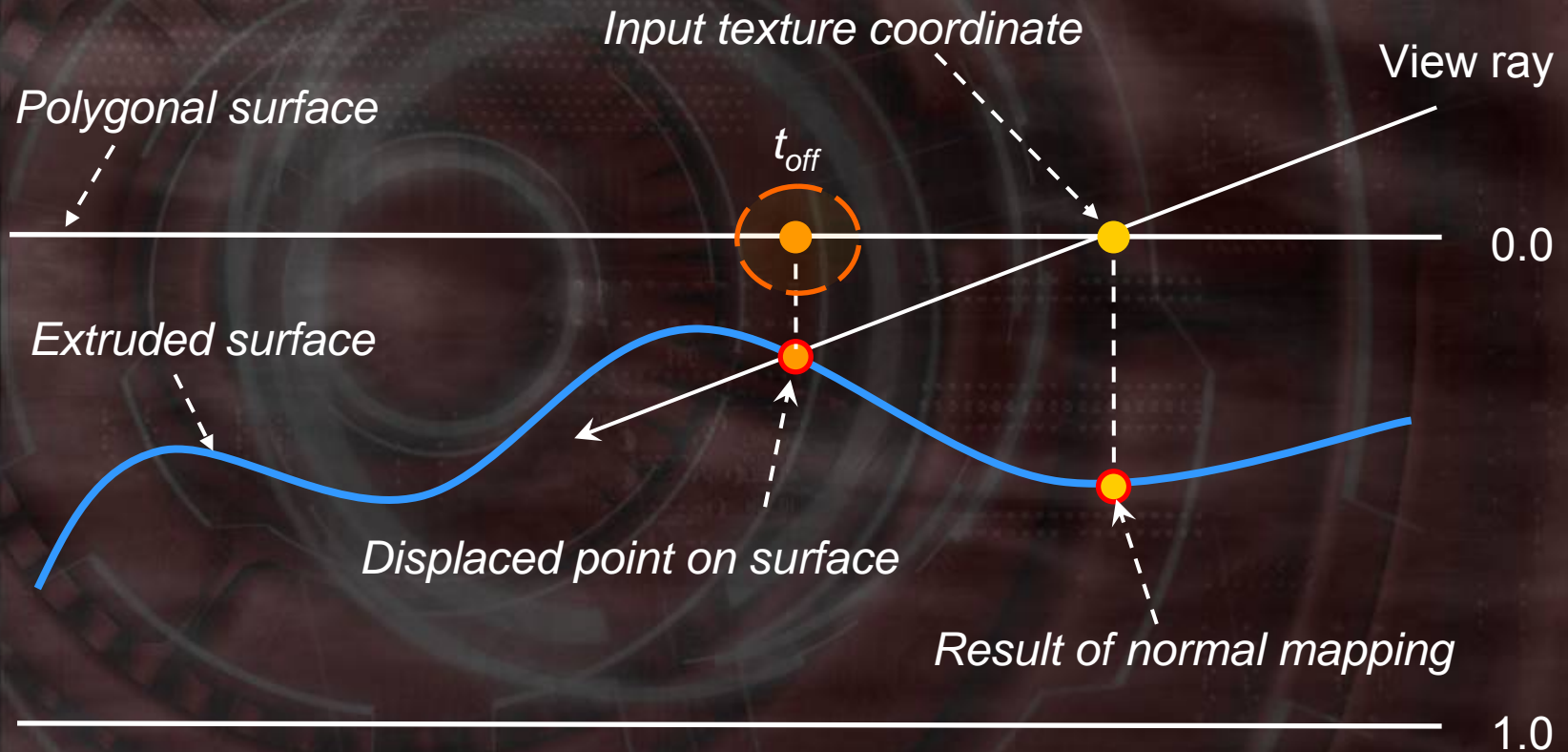
# Parallax Occlusion Mapping



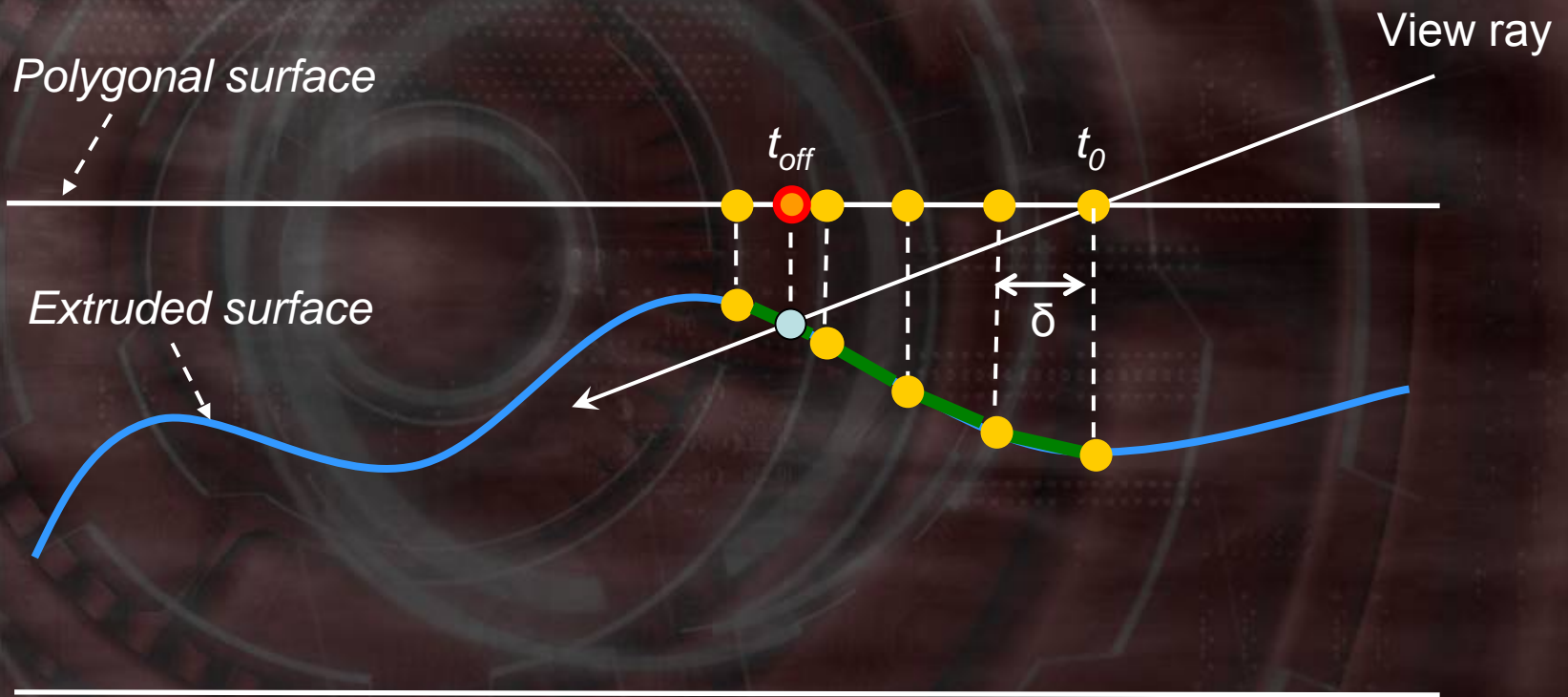
- Per-pixel ray tracing at its core
- Correctly handles complicated viewing phenomena and surface details
  - Displays motion parallax
  - Renders complex geometric surfaces such as displaced text / sharp objects
  - Uses occlusion mapping to determine visibility for surface features (resulting in correct self-shadowing)
  - Uses flexible lighting model



# Parallax Displacement



# Height Field Profile Tracing



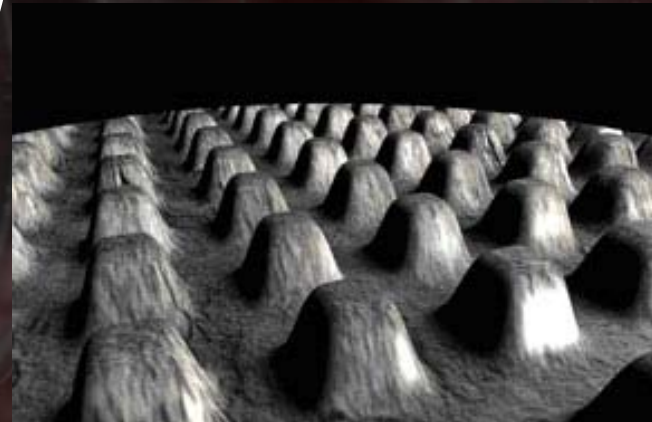
# Dynamic Sampling Rate



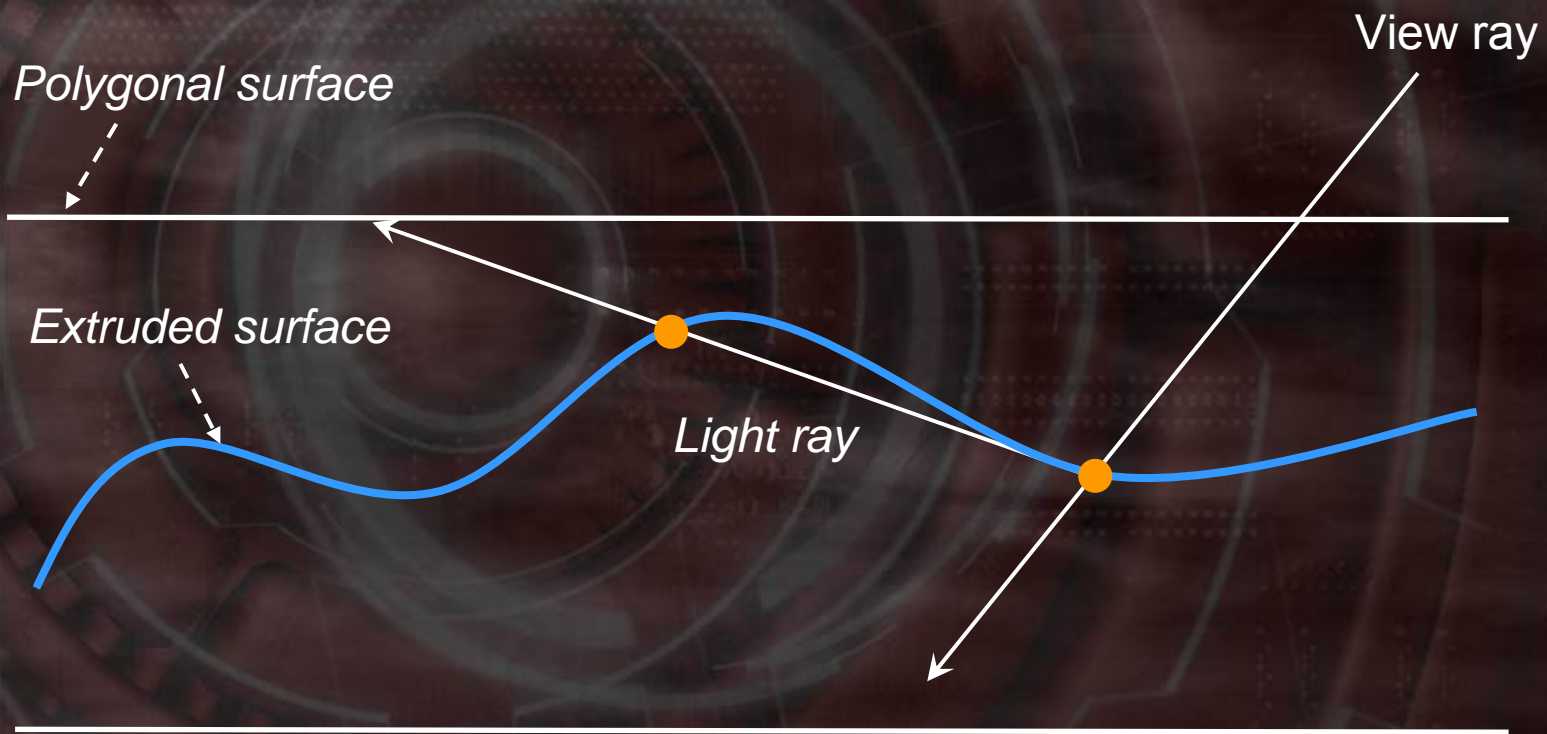
- Sampling-based algorithms are prone to aliasing
- The optimal sampling rate is view dependant and geometry dependant
  - We compute this using the viewing vector and the surface normal.
- Dynamic flow control in the pixel shader (a key SM 3.0 feature) allows us to dynamically adjust the sample rate per pixel.

*Aliasing at grazing angles due to static sampling rate*

*Perspective-correct depth with dynamic sampling rate*



# Self-Occlusion Shadows

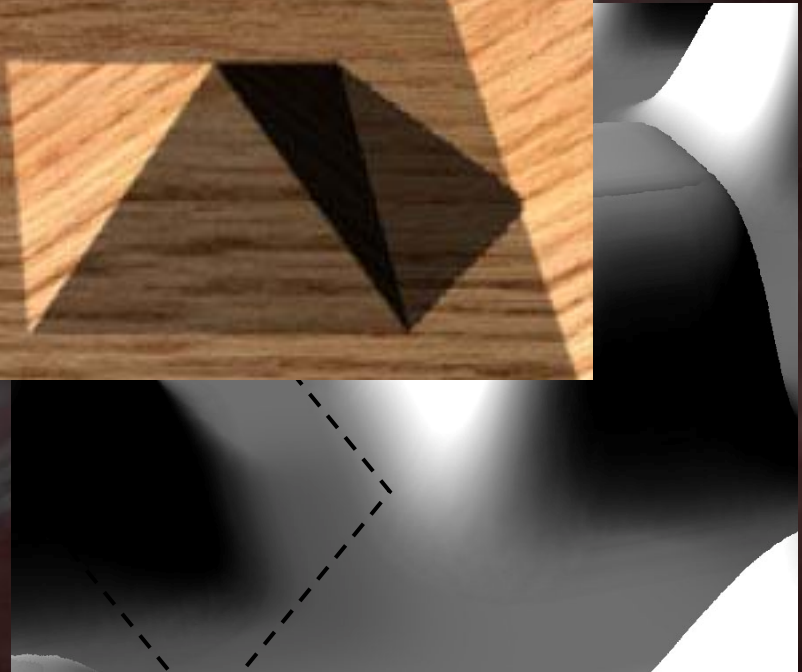


# Soft Shadows Computation



- Simply determining whether the current feature is occluded yields hard shadows
- We can compute soft shadows by filtering the visibility samples during the occlusion computation
- Don't compute shadows for objects not facing the light

$$N \cdot L > 0$$



# Adaptive Level-of-Detail System



- Compute the current mip map level
  - Using SM3.0 gradient computation instructions
- For furthest LOD levels, render using normal mapping
- As the surface approaches the viewer, increase the sampling rate as a function of the current mip map level
- In transition region, blend between the normal mapping and the full parallax occlusion mapping



# Parallax Occlusion Mapping Demo

# Rendering Realistic Rain



- Simply rendering particle rain is not enough
  - Rain has a variety of visual cues
  - Missing pieces can destroy the illusion of immersion
- We developed and combined a number of novel techniques to create this natural scene of a rainy city street at night



*Rainy night in Boston*



*Rainy night in Toy Shop Town*



# Rendering Realistic Rain

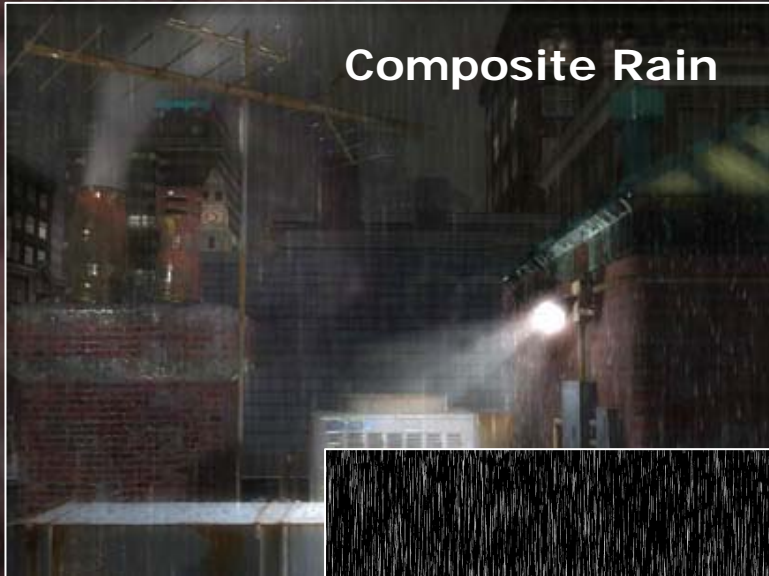


- Techniques we employ
  - Falling Rain through composite layer rain technique
  - Wet Surfaces
  - Raindrop splashes due to collisions with surfaces
  - Raindrops falling from edges
  - Water Puddles with ripples, reflections & refraction
  - Running Water
  - Misty objects outlines
  - Droplets & Streaks on windows
  - Reflections
  - Smears on Car Windshield
  - Glow for bright objects due to light refraction in the air
  - Fog
- Over 300 unique shaders used - just for the rain effects.

# Creating the Illusion of Rain



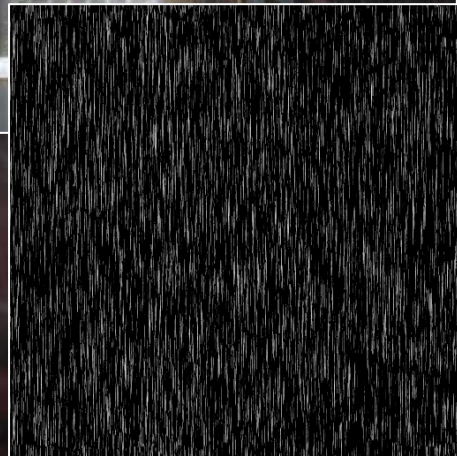
Composite Rain



Raindrop splashes



Raindrops off objects



# Creating the Illusion of Rain



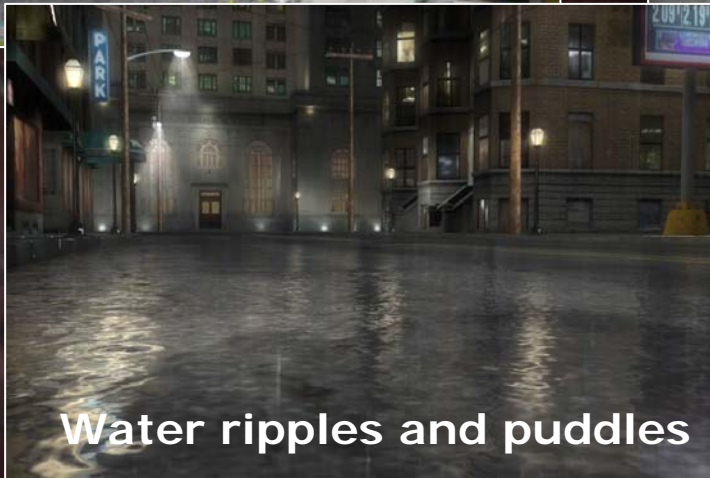
Water droplets on windows



Misty objects in the rain



Water ripples and puddles



Streaming water



# Creating the Illusion of Rain



Fog and Glow

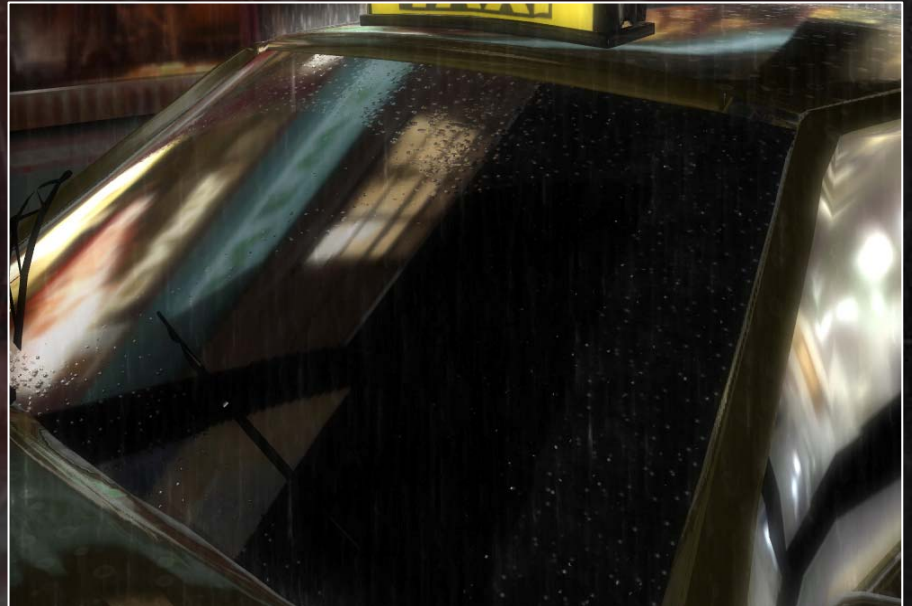


Wet reflections

# 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



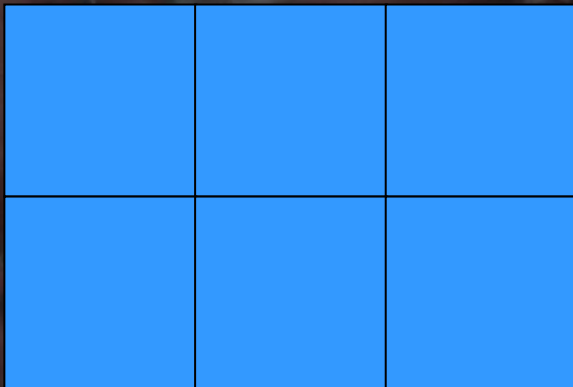
# Droplet Simulation



# Droplet Simulation



- Modification of "Animation of water droplets moving down a surface" [Kaneda99].
- 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 within the cell.
- Convenient to pack into .rgba 16-bit per channel texture.

# Droplet Simulation



$$V_{new} = V_i + F/M * t$$

$$F_{down} = M * G$$

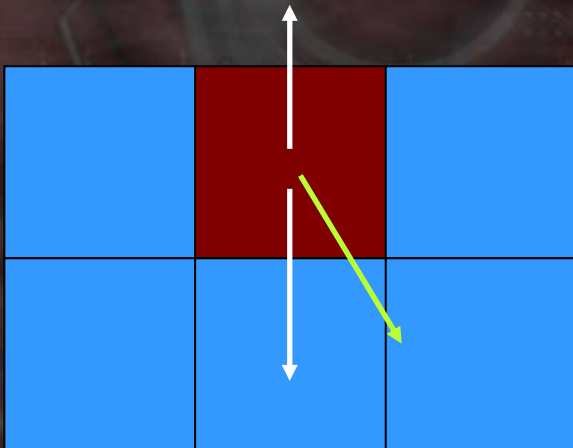
$$F_{up} = f_{static} \text{ or } f_{dynamic}$$

- Gravity and mass are used to compute the downward force on the droplet.

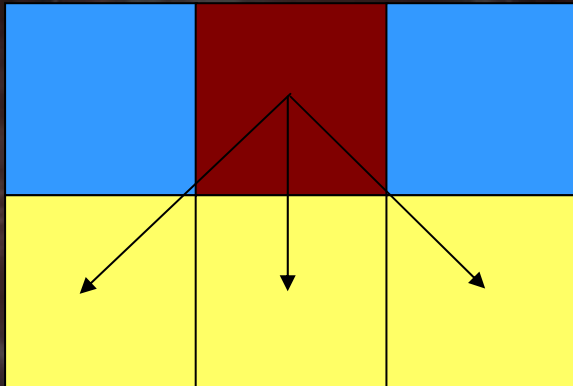
- Static friction for stationary droplets, and dynamic friction for moving droplets is used to compute the competing upward force.

- 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.



# Droplet Simulation

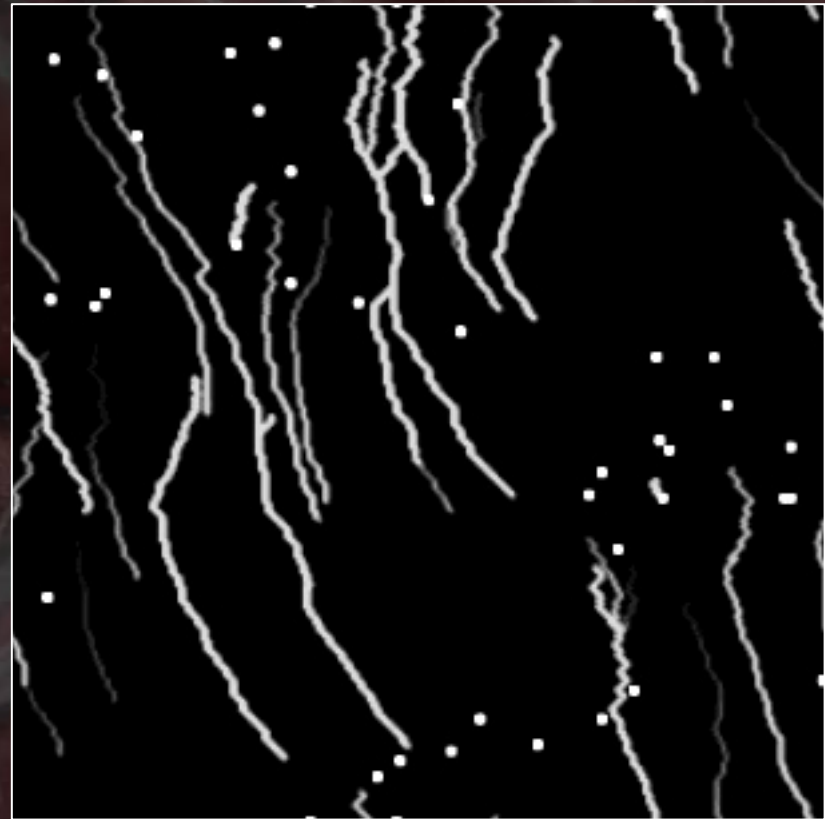


- Droplets can flow into one of the 3 cells below.
- New cell to flow into is randomly chosen biased by velocity, friction based affinity and wetness of the target cell.
- Droplets have a greater affinity for wet regions of the surface.
- Velocity is updated based on target cell chosen

# Droplet Simulation Rendering



- After simulation, each cell contains a new mass value.
- A bumpmap 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 toystore.





# Droplet Simulation Demo

# Dynamic Water Ripples in Puddles



- Water ripples are generated as a result of rain drops falling onto the geometry in the scene
  - Can be a direct response
  - In our case, we approximate by seeding rain drops into a texture
- 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
- Simulate realistic wave motion interactively over the water surface

# Our Approach



- Approximate water surface with a lattice of points
    - We render our surfaces with a 256 x 256 simulation
    - Each lattice contains information about water surface at that point
      - Position
      - Velocity
      - Acceleration
- } Packs nicely into a single RGB texture
- Similar to "Interactive Simulation of Water Surfaces" by M. Gomez (*Game Programming Gems*)

# Our Approach



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

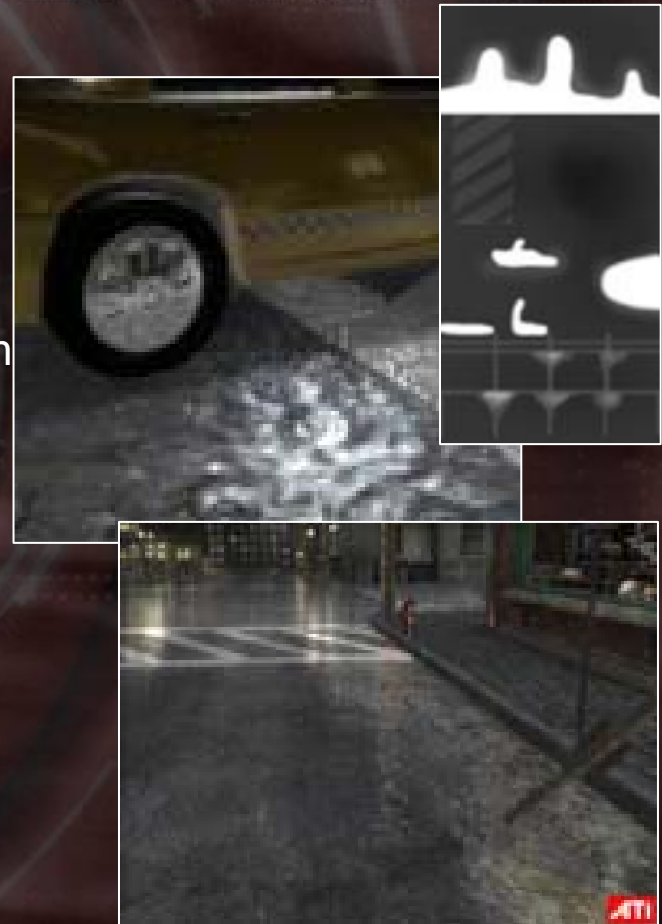
$$\frac{\partial^2 z}{\partial t^2} = v^2 \left( \frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} \right)$$

- Where  $v$  is the velocity of the waves traveling across the surface
- We solve this equation in real-time in pixel shaders to determine water waves height for each point in the lattice

# Rendering Water Puddles



- Artist can specify a puddle mask
  - Placement of puddles on geometry
  - Also specifies puddle depth variation
- Combine water surfaces with the underlying geometry
  - Perturb bump map normals of the main geometry with the water surface normals
  - Render both shallow puddles and deeper puddles as well as just wet street
    - Use the puddle depth map
- No additional geometry is required for water puddles.
- Demo



# Toy Shop - More Details



- Lightning lightmaps
  - Illumination due to lighting flashes is integrated into every object.
  - Uses two single channel lightning lightmaps.
  - Uses 3DC+ compression
- Swinging telephone lines
  - Uses catenary equations to compute the vertex positions for the drooping of the cables.
  - Artist controllable.
- Shadow Mapping using DF16 depth texturing.
  - Memory requirements were critical for this demo. DF16 saves memory and bandwidth.

# Conclusion



- Generating content for next generation platforms is a major challenge for developers.
  - Can only be solved by programmers and artists working together.
  - Key Techniques Include
    - Capture from real-world data
    - Procedural animation for secondary effects
    - Artist-directable Shaders
- Dynamic Flow Control in SM 3.0 helps with detail management.
  - Has great performance on ATI hardware