GPU-Assisted Rendering Techniques

Matthias M Wloka
Shader Model 3.0 Hardware Is Here!

• What can I do with it?

• How can it help me with my game?
  – How can I use it to offload the CPU?
  – How can it help my image quality?
How Can It Help My Game?

• Offload work from the CPU to the GPU
  – “Render To Vertex Buffer” technique
  – Uses texture-fetch feature of VS 3.0

• Image quality
  – Per-pixel specular lighting
  – Normal-map mipmap maps produce artifacts (distant objects shimmer)
Offloading the CPU

Typical Workflow

1. Simulate
2. Vertex positions
3. Render to frame-buffer
Offloading the CPU

Typical Workflow

1. **Simulate**
2. **Vertex positions**
3. **Render**
4. **GPU**
   - to frame-buffer
Simulating On the GPU?

- Use them programmable shaders!

![Diagram showing the process of simulating on the GPU and rendering to a frame buffer with high bandwidth.](image-url)
“Render To Vertex Buffer”

- Removes read-back from GPU to CPU

- Render to texture
- Store vertices as texture data
- Read texture into vertex shader
- Simulate
- Texture
- Render to frame-buffer

GPU

~10s GB/s
Examples

• Cloth
  – Collide cloth against scene
  – Run cloth physics: damped springs

• Displacement Mapping
  – Displace vertices
More Examples

• Snow/Sand accumulation
  – Simulate friction/sliding

• Wind (simulation) bending grass

• Particle Systems

• Water waves/wakes
Rendering Water

- Tessellated, flat plane for water

1. Solve wave-equations
2. Store vertex heights
3. Read height in vertex shader
4. Render to frame-buffer
How Does It Work?

• Create a vertex-mesh for water surface
  – Say, 128 by 128 vertices
  – Encode vertex’s mesh-position as uv-coordinates
Vertex Shader Work

• Read ‘height-map’
  – Floating-point texture
  – Read texture at vertex’s uv

• Add result to vertex’s y

• Transform/Project vertex

\[ \text{vertex.y} += \text{tex}(u, v) \]
\[ \text{Out.pos} = \text{WorldViewProj} \times \text{vertex} \]
Height-Map Is Dynamic

- Update every frame
  - With GPU via render-to-texture

- Simulate water movement with Verlet integration
Verlet Integration

• \( A = \sum \text{(neighbors)} - 4 \ H_{n-1} \)
  \( H_n = (H_{n-1} - H_{n-2}) + A \)
  – Operates on positions only
  – No need to store velocity or acceleration

• Compute normal from positions:  
  \( N = \text{Normalize}(S \times T) \)
Add Disturbances to Height Map

• Blend displacements into the water
  – For example: the boat, rocks, shore

• Verlet-integration integrates it next frame

• Yes, floating-point render-target blending
Add Usual Eye Candy

• Caustics
  – Bilinear filtering of the normals crucial: low-res (128x128) texture

• Reflection/Refraction

• Fresnel
Advantages

• Fast!
  – Simulation happens on 128x128 texture
  – Small by GPU standards
  – Frame-rate unaffected by simulation

• Reasonable geometric complexity
  – 128x128 is 16k vertices
More Details on This Sample (and Others)

- Next-Gen Special Effects Showcase
  - Wednesday, 12-1pm
Particles via Render-to-VB

- Building a Million Particle System
  - Lutz Latta
  - Wednesday, 12-1pm
Image Quality

• Per-pixel specular lighting
  – Normal-map mipmaps produce artifacts (shimmer on distance objects)
  – Uses floating-point texture filter and blend
Normal-Map Mipmap Artifacts
What Is the Problem?

- Specular term \((N \cdot H)^s\) is high-frequency function:
Sampling Frequencies

- Magnification case: accesses top-level mipmap
  - Sufficient sampling

- Minification case: lower mip-levels
  - Without mipmaps: sparkle city
  - With mipmaps: better, but not much
What Is a Normal-Map Mipmap?

- Averaging
  - Replace 4 normals with 1 completely different normal

- Not re-normalizing shortens that normal
  - Scales down dot-product
What We Really Want

• $N \cdot H$ is a hack
  – $N$ represents all normals in texel

• Integrate over all normals in texel
  – Integral is $N \cdot H$ only if all $N$ in texel are the same
  – Not true for mipmaps
How To Integrate

• Approximate dot-product via Gaussian:
  \[(N \cdot H)^s = \cos^s \alpha \approx e^{-\frac{1}{2} s \alpha^2}\]

• Gaussians with standard deviation
  – Sum of them is another Gaussian with standard deviation
After A Lot Of Math…

• Dot-product with Gaussian is:
  \[ N_a = \sum_i N_i \]
  \[ f_t = \frac{1}{1 + s \left( \frac{1}{|N_a|} - 1 \right)} \]  (Toksvig factor)

specular term = \( f_t \left( \frac{(N_a \cdot H)}{|N_a|} \right)^{f_t s} \)
Corner Cases

• Dot-product with Gaussian is:
  \[ N_a = N \]
  \[ f_t = \frac{1}{1 + s \left( \frac{1}{|N_a|} - 1 \right)} \]  (Toksvig factor)

specular term = \( f_t \left( \frac{(N_a \cdot H)}{|N_a|} \right)^{f_t s} \)
Corner Cases

• Dot-product with Gaussian is:
  \[ N_a = N \]
  \[ f_t = \frac{1}{1 + s \left( \frac{1}{|N|} - 1 \right)} = 1 \]

  specular term = \[ 1 \left( \frac{(N \cdot H)}{|N|} \right)^1 \]
  = \[ (N \cdot H) \]
Another Corner Case

• Dot-product with Gaussian is:
  \[ N_a = \sum_i N_i = 0 \]
  \[ f_t = \frac{1}{1 + s (1/0 - 1)} = 0 \]

  specular term = 0 \( (0 \cdot H) / 0 \)^
  ^{0 \cdot s}
Effect

• Length of normal expresses distribution

• Constant normal across texel
  – Computes sharp high-lights

• Widely varying normals across texel
  – High-light faints and widens
Messy Formula To Compute?

- Function $f_t \left( \frac{(N_a \cdot H)}{|N_a|} \right)^s$ depends on
  - $s$ (constant)
  - $N_a \cdot H$
  - $N_a \cdot N_a$

- 2D texture look-up: $\text{tex}(N_a \cdot H, N_a \cdot N_a)$
  - Different 2D textures for different $s$
Implementation

• Generate mipmaps via averaging
  – Leave vectors un-normalized!

• Fetch $N_a$
  – Fp16 to minimize precision errors
  – Anisotropic filtering for best results
Implementation Continued

• Compute $N_a \cdot H$ and $N_a \cdot N_a$

• Fetch specular using those coordinates
  – Since it is a function look-up:
  – Bilinear only
  – No mipmaps
  – 128x128 works ok
Result and Observations

• Short normals in base-level: reduced specularity
• Specialize normal mipmap generation
• Applies to interp. vertex normals
• Applies to reflection-map lookups (LOD them)
Before/After Comparison

Before

After
Questions, Comments, Feedback?

• mwloka@nvidia.com

• http://developer.nvidia.com
  The Source for GPU Programming

• Slides available online
More Rendering Techniques: NVIDIA’s SDK 7.0

- 200+ rendering techniques
- CD available @ NVIDIA’s booth
- Free
Other Rendering Technique Talks

• Cinematic Effects II: The Revenge
  – Wed, 9-10am

• GPU Gems Showcase
  – Wed, 5:30-6:30pm

• Real-Time Translucent Animated Objects
  – Fri, 2:30-3:30pm