Next Generation Shading and Rendering

Bryan Dudash
NVIDIA
Session Overview

3.0 Shader Model Overview
- ps.3.0 vs. ps.2.0
- vs.3.0 vs. vs.2.0

Next-Gen Rendering Examples
- Dynamic Water
  - Vertex Texture Fetch
  - Floating-point filtering / blending
  - GPU-based physics simulation
- Volumetric Fog
  - MRT and branching for speed
- Deferred Rendering
  - MRT and branching for speed

Geometry Instancing
- Added visual complexity
- Performance optimization
# Pixel Shader 3.0 Feature Comparison

<table>
<thead>
<tr>
<th>Pixel Shader Feature</th>
<th>Shader 2.0</th>
<th>Shader 3.0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shader length</td>
<td>96</td>
<td>65535+</td>
<td>Allows more complex shading, lighting, and procedural materials</td>
</tr>
<tr>
<td>Dynamic branching</td>
<td>No</td>
<td>Yes</td>
<td>Saves performance by skipping complex shading on irrelevant pixels</td>
</tr>
<tr>
<td>Shader anti-aliasing</td>
<td>Not supported</td>
<td>Built-in derivative instructions</td>
<td>Developers can calculate the screen space derivatives of any function, allowing them to adjust shading frequencies or over-sampling to eliminate artifacts</td>
</tr>
<tr>
<td>Minimum Precision</td>
<td>fp24</td>
<td>fp32</td>
<td>Fewer artifacts, more dynamic range</td>
</tr>
<tr>
<td>Back-face register</td>
<td>No</td>
<td>Yes</td>
<td>Allows two-sided lighting in a single pass</td>
</tr>
<tr>
<td>Pixel Shader Feature</td>
<td>Shader 2.0</td>
<td>Shader 3.0</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Back-face register</td>
<td>No</td>
<td>Yes</td>
<td>Allows two-sided lighting in a single pass</td>
</tr>
<tr>
<td>Interpolated color format</td>
<td>8-bit integer minimum</td>
<td>32-bit floating point minimum</td>
<td>Higher range and precision color allows high-dynamic range lighting at the vertex level</td>
</tr>
<tr>
<td>Multiple render targets</td>
<td>Optional</td>
<td>4 required</td>
<td>Allows advanced lighting algorithms to save filtering and vertex work – thus more lights for minimal cost</td>
</tr>
<tr>
<td>Fog and specular</td>
<td>8-bit fixed function minimum</td>
<td>Custom fp16-fp32 shader program</td>
<td>Shader Model 3.0 gives developers full and precise control over specular and fog computations, previously fixed-function</td>
</tr>
<tr>
<td>Texture coordinate count</td>
<td>8</td>
<td>10</td>
<td>More per-pixel inputs allows more realistic rendering, especially for skin</td>
</tr>
</tbody>
</table>
## Vertex Shader 3.0 Feature Comparison

<table>
<thead>
<tr>
<th>Vertex shader feature</th>
<th>Shader 2.0</th>
<th>Shader 3.0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shader length</td>
<td>256 Instructions</td>
<td>65535 instructions</td>
<td>More instructions allow more detailed character lighting and animation</td>
</tr>
<tr>
<td>Dynamic branching</td>
<td>No</td>
<td>Yes</td>
<td>Saves performance by skipping animation and calculations on irrelevant vertices</td>
</tr>
<tr>
<td>Vertex texture</td>
<td>No</td>
<td>Any number of lookups from up to 4 textures</td>
<td>Allows displacement mapping, particle effects</td>
</tr>
<tr>
<td>Instancing support</td>
<td>No</td>
<td>Required</td>
<td>Allows many varied objects to be drawn with only a single command</td>
</tr>
</tbody>
</table>

©2004 NVIDIA Corporation. All rights reserved.
So what can we do with all this?

- Dynamic Water Rendering
  - VS3.0 Vertex Texture Fetch
  - Floating-point filtering / blending
  - GPU-based physics simulation

- Animated Volumetric Fog
  - Use polygon primitives to bound fog
  - MRT and branching for speed

- Geometry Instancing
  - Draw many “instances” of a mesh with one draw call
Typical Workflow

Simulate → Vertex positions → Render → to frame-buffer

©2004 NVIDIA Corporation. All rights reserved.
Typical Processing Allocation

Simulate ➔ Vertex positions ➔ Render ➔ to frame-buffer

CPU ➔ GPU
Simulating On the GPU?

Use them programmable shaders!
The read-back can kill you
This is for PCI. PCI Express is better.

Read-back: BAD!

Simulate $ightarrow$ Vertex positions $ightarrow$ Render

Simulate

Vertex positions

Render to frame-buffer

GPU

CPU

GPU

~100s MB/s
“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
- 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 – Algorithm Overview

- Perform water simulation in pixel shader
  - Render to texture
    (D3DFMT_A16B16G16R16F)

- Render refraction and reflection maps

- Render water surface
  - Use simulation results via VS3.0 vertex texture fetch
  - Compute perturbed texture coordinates
  - Combine refraction and reflection using Fresnel term