Procedural Shaders: A Feature Animation Perspective

Hector Yee, Rendering Specialist, PDI/DreamWorks
David Hart, Senior FX Developer, PDI/DreamWorks
Arcot Preetham, Engineer, ATI Research

Motivation

- Movies still look better
- Up visual bar with programmable graphics hardware
- Borrow techniques from Feature Animation for use in Real Time
Motivation – get from here

Jak 2
(2003)
PS 2
Naughty Dog

... to here

Shrek 4D
(2003)
Film
PDI/
DreamWorks
Talk Outline

- Technological similarities & differences
- Techniques from feature animation
- Techniques from real-time rendering

Where we are

- Typical values for Shrek
  - Typical frame
  - Pentium 4 @ 2.8 GHz
- Typical values for DX 9 part
  - Assuming 30 FPS
  - Based on Radeon 9800
  - Some values based on theoretical max
“Typical” Shrek frame

Similarities

<table>
<thead>
<tr>
<th>Technology</th>
<th>Feature Animation</th>
<th>Realtime Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>720 x 486 (NTSC)</td>
<td>640 x 480</td>
</tr>
<tr>
<td></td>
<td>1828 x 1102 (Academy 1.66)</td>
<td>1024 x 768</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1280 x 1024</td>
</tr>
<tr>
<td>Anti-Aliasing</td>
<td>8 x 8</td>
<td>4 x 4</td>
</tr>
<tr>
<td>Bits per channel</td>
<td>32 (internal float)</td>
<td>32 (internal float)</td>
</tr>
<tr>
<td></td>
<td>4-8 (YUV 4:2:2)</td>
<td>8 (RGB 8:8:8)</td>
</tr>
</tbody>
</table>
Differences (Geometry)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Feature Animation</th>
<th>Realtime Rendering</th>
<th>Order of Magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time per frame</td>
<td>8000 secs</td>
<td>0.015 secs</td>
<td>6</td>
</tr>
<tr>
<td>Polys / frame</td>
<td>100 M</td>
<td>0.1M - 1M</td>
<td>2</td>
</tr>
<tr>
<td>Bones &amp; Skinning</td>
<td>350 CPU proc.</td>
<td>32</td>
<td>1</td>
</tr>
</tbody>
</table>

Geometric Resolution

- Feature Animation
  - Mostly procedural geometry
  - NURBS, NUBS or subdivision surfaces
- Realtime
  - Usually triangles and quads
  - Recently N-patches or RT-patches
### Differences (Rendering)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Feature Animation</th>
<th>Realtime Rendering</th>
<th>Order of Magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time per frame</td>
<td>7000 secs</td>
<td>0.015 secs</td>
<td>6</td>
</tr>
<tr>
<td>Number of Lights</td>
<td>100</td>
<td>5 - 10</td>
<td>2</td>
</tr>
<tr>
<td>Shadow samples</td>
<td>1000 (soft shadows)</td>
<td>1 (depth map)</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Shader network for Shrek’s body**

*Realtime Rendering Feature Animation Technology*
Differences (Shading)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Feature Animation</th>
<th>Realtime Rendering</th>
<th>Order of Magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shader ops per pixel</td>
<td>1 M</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>Shader Parameters</td>
<td>~100 (chained)</td>
<td>~10</td>
<td>2</td>
</tr>
<tr>
<td>Texture RAM</td>
<td>1545 MB</td>
<td>64 MB</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Other Differences

- Texture Filtering
  - Analytic vs Trilinear Mipmap (Dave)
- Shader Environment
  - P, dPdUV, N vs streams (Dave)
- Shading Language
  - C/C++ vs Cg/HLSL/GLSL (Preetham)
- Color Calibration
Color Calibration

- Consistent view for
  - Artists, content provider, consumers
- Feature Animation
  - Artists calibrate, Theatres calibrate
- Realtime Rendering
  - Artists calibrate (sometimes)
  - Gamers turn up the gamma!

Shader Environment

By *shader* I mean *plugin*

Compiled .dso (.dll) written in C

Materials, maps, lights, geometry, etc..

Shaders are (ideally) stream filters / DG nodes

Look at inputs and outputs only

But we (PDI) always cheat

Traversing scene, loading files, ray tracing, etc..

Full access to all app. libraries
Shader Environment

P, N, Ng, UV, dPd[UV], ref[PN], etc...

These data come in both singles & *tuples*

- Singles = data at the poly center
- Tuples = data at poly vertices
  
  (e.g. vertex normals, vertex UVs, etc...)
Anti-aliasing

No-one *wants* aliasing, but in reality...

Hardware support
Performance
features / quality / speed

no aliasing allowed (noise is not OK)

Fortunately, we (FA) have lots of time

---

Image mapping for RT

Input UV is a single

Tri-linear MIPMAP interpolation

MIPMAP is point-sampled using single (face) UV
Image mapping for FA

Input UV is tuple

Integrate filtered
texels in tuple

Quality knob chooses MIPMAP level
(e.g. GL_TEXTURE_LOD_BIAS_EXT in openGL)

Brick Shader

Use uv tuple polygon

Find fully & partly
enclosed bricks

Fully enclosed =
average color

Partly enclosed =
clip & evaluate
Brick Shader

Function maps $(I, N)$ to UV
Using reflection vector $R$
Builds on Image mapping
Tuple UVs computed with tuple $N \ & P$
UV tuple is passed on to image map
Env mapping

Tuple UVs might cross seams, so subdivide tuple UV polygon
Each tuple polygon is evaluated by image map shader.
Env mapping

Procedural noise

We use noise heavily
Many different types
  gradient, cell, convolution,
  turbulence, marble, worley
  1d, 2d, 3d, 4d...
Fractal noise anti-aliasing
  Evaluate frequency in ‘octaves’
  Only evaluate the frequencies
  that are below Nyquist limit
Shading Models

Wide range of complex models
Default material has standard terms:
  Ambient, Diffuse, Specular
And some non-standard terms:
  Shadowing, Directional ambient, Directional diffuse,
  Retro-reflection, Fresnel reflectivity, transparency...
Not just surface materials:
  Maps, Fabric, Fur, Particles, Volumes,

Fur Shader

Shading model for curves [Kajiya ’89]
Shrek 4D
Shrek 4D

Shading Languages

CPU

- RenderMan®
- C Libraries

GPU

- HLSL, GLSL, Cg.
- Assembly.
Shading Blocks

- On CPU
  - Light, Surface, Volume shaders.
- On GPU
  - Vertex & Pixel shaders.

Shading on graphics hardware

- Instruction set
  - Limited Control Flow
  - No Bitwise operators
- Resources
  - Limited Registers (Temp, Interpolators, Constants)
- No Global Memory
Shading on graphics hardware (cont’d)

• Finite number of instructions

![PS Shader Model vs Instructions](chart)

Multipassing

• Interactive multi-pass programmable shading, Siggraph 2000 - Peercy et al

• Efficient partitioning of fragment shaders for multi-pass rendering on programmable graphics hardware, Siggraph 2002 – Chan et al.
CPU vs GPU Shading

- Use GPU for offline shaders.
  - Procedural Lights.
  - Complex Surfaces.
  - Noise.
Lights

- Fixed Function
  - 8 lights
  - Dir, Point, Spot
- Programmable
  - Any number of lights
  - Custom light shaders
  - Eg. Windowlight

Windowlight

- Light through a window.
- Parameters:
  - # horiz panes, vert panes
  - from, to, up
  - frame width & height
  - fuzz.
Surface Models

- Fixed Function
  - Diffuse, Phone, Multi-texture
- Programmable
  - Custom surface models.
  - Eg. OrenNayar, Anisotropic, Fur.

Fur

- Fur geometry rendered as triangles.
- Shading uses fur tangent direction
Noise

- Widely used in studios.
- GLSL & HLSL shading languages have noise functions.
- Popular implementation
  - Perlin noise
- Eg: Ocean waves

Texture Noise

- Noise based demos used textures.
- Advantages: Fast.
- Disadvantages: Repeat, memory expensive, linear filtering
Procedural Noise

- Advantages: No filtering artifacts.
- Disadvantages: Computationally expensive.
- Perlin noise implementation on GPU.
  - float noise3d(): 56 alu, 16 tex.
  - float3 noise3d(): 172 alu, 48 tex.

Conclusion

Cinematic quality in real time?
Still a long way to go.
Acknowledgements

PDI: Jonathan Gibbs, Jerry Hebert, Harry Max, Paul Rademacher, Dan Yu, Taylor Shaw, Andy Hendrickson, Rachel Falk, lighting, char TDs, FX, R&D

ATI: Avi Bleiweiss, Dominik Behr, Seth Sowerby, Pierre Boudier, Axel Mamode, Mike Huber, Raja Koduri.