Procedural Shaders: A Feature Animation Perspective

Hector Yee, Rendering Specialist, PDI/DreamWorks
David Hart, Senior FX Developer, PDI/DreamWorks
Arcot J. Preetham, Senior 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
Shrek 4D (2003) Film
PDI/DreamWorks
Talk Outline

• Technological similarities & differences

• Techniques from feature animation

• Techniques from real-time rendering
Technological Aspects

- Outline of pipelines
- ‘Typical’ scenes and values
- Similarities
- Differences
  - Pipeline, Geometry, Rendering, Shading
  - Other
Programmable Pipeline

- **Vertex operations**
  - Shade & transform vertices
  - Interpolate vertex data
- **Pixel operations**
  - Shade pixels
  - Discard (Alpha, Z, stencil, clip)
  - Output fragment color
  - Frame buffer blends
Feature Animation Pipeline

- Generate geometry
- Tessellate, displace, dice
- Discard (visibility)
- Shade micro-polygon
- Filter micro-polygons
- Frame buffer operations
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
Shrek 4D
## 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) 1828 x 1102 (Academy 1.66)</td>
<td>640 x 480 1024 x 768 1280 x 1024</td>
</tr>
<tr>
<td>Bits per channel</td>
<td>32 (internal float) 4-8 (YUV 4:2:2)</td>
<td>32 (internal float) 8 (RGB 8:8:8)</td>
</tr>
</tbody>
</table>
### Differences (Pipeline)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Feature Animation</th>
<th>Realtime Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixels per Polygon</td>
<td>~0.1 (!)</td>
<td>~25</td>
</tr>
<tr>
<td>Visibility</td>
<td>Z-test before shading</td>
<td>Z-test after vertex shading</td>
</tr>
<tr>
<td>Shading Interpolation</td>
<td>Constant shaded micro-polygons</td>
<td>Interpolate vertex shading</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>4 mat/bone</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>Anti-aliasing</td>
<td>64 QMC</td>
<td>1 – 16 r. grid</td>
<td>~1 - 2</td>
</tr>
<tr>
<td>Shadow samples ppix</td>
<td>1000 (soft shadows)</td>
<td>1 (depth map)</td>
<td>3</td>
</tr>
</tbody>
</table>
Differences (Shading)

Shader network for Shrek’s body
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</td>
<td>~10</td>
<td>1</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 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...)
Polygon

Texture
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
Env mapping

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

Singles eval

Tuples eval
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
Shrek 4D
Shrek 4D
Shrek 4D
Shrek 4D
Shrek 4D
Shading Languages

- **Movies**
  - RenderMan ® Shading Language
  - C/C++ Libraries
- **Games**
  - Assembly Language
  - HLSL, GLSL
Shading Blocks

- Offline Shaders
  - Light, Surface, Volume, Displacement shaders.

- Real-time Shaders
  - Vertex & Pixel shaders.
Shading on Graphics Hardware

• Instruction set
  – Limited Control Flow
  – No Bitwise operators

• Resources
  – Limited Registers (Temps, Interpolators, Constants)
Shading on graphics hardware (cont’d)

- Finite number of instructions
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.
Offline vs Real-time Shading

- Goals are completely different.
  - Movies: Quality.
  - Games: Speed.
Noise

- Widely used in movies.
- Popular implementation
  - Perlin noise
- GLSL & HLSL shading languages have noise functions.
Procedural Noise

• Disadvantages: Computationally expensive.
• Advantages: Smooth function.
• Perlin noise implementation on GPU.
  – float noise(vector ) - 56 alu, 16 tex.
  – color noise(vector ) - 172 alu, 48 tex.
Texture Noise

- Real time demos use textures for noise.
- Advantages: Fast.
- Disadvantages: Repeat, memory expensive, linear filtering.
Worley Noise


![Diagram showing the process of Worley Noise]

- Jitter feature points
- Find nearest feature point
void voronoi_f1f2(point P;  float jitter;
    output float f1;  output point pos1; output float f2;  output point pos2; )
{
    point thiscell = point (floor(xcomp(P))+0.5,floor(ycomp(P))+0.5,floor(zcomp(P))+0.5);
    f1 = f2 = 1000;
    float i, j;
    for (i = -1;  i <= 1;  i += 1) {
        for (j = -1;  j <= 1;  j += 1) {
            point testcell = thiscell + vector(i,j,0);
            point pos = testcell + jitter(P) * (vector cellnoise (testcell) - 0.5);
            vector offset = pos - P;
            float dist = offset . offset; /* actually dist^2 */
            if (dist < f1) {
                f2 = f1;  pos2 = pos1;
                f1 = dist;  pos1 = pos;
            } else if (dist < f2) {
                f2 = dist;  pos2 = pos;
            }
        }
    }
    f1 = sqrt(f1);  f2 = sqrt(f2);
}
Worley Shader (cont’d)

```
surface Foo()
{
    float f1, f2;
    point pos1, pos2;
    point T = point(s, t, 0);
    voronoi_flf2(T, 0.5, f1, pos1, f2, pos2);
    Ci = color cellnoise(pos1);
}
```

- Translates to ~150 pixel shader instructions.
Worley Shader (cont’d)

surface Foo()
{
    float f1, f2;
    point pos1, pos2;
    point T = point(s, t, 0);
    voronoi_f1f2(T, 0.5, f1, pos1, f2, pos2);
    Ci = color(f1);
}

- Translates to ~135 pixel shader instructions.
Worley Shader (cont’d)

```glsl
surface Foo()
{
  float f1, f2;
  point pos1, pos2;
  point T = point(s, t, 0);
  voronoi_f1f2(T, 0.5, f1, pos1, f2, pos2);
  Ci = color(step(0.1, f2-f1));
}
```

- Translates to ~190 pixel shader instructions.
Worley Noise Demo
Fur

Screen shot from movie “Shrek”
Fur Geometry

• Vertex data
  – position, tangent, radius, tex coordinate, base position.

• Parameters
  - width & length scale.
Fur Shading

Feature animation shader

Kajiya’s lighting model
(Siggraph 1989)

User Interface
Indirect Lighting
Shadowing
Translucency

Real time shader
void main()
{
  // Incident direction.
  vec3 I = Inc;

  /* Hair direction */
  vec3 hair_direction = normalize(Nor);

  /* Accumulate color */
  vec4 C = vec4(0, 0, 0, 0);

  /* ambient */
  C = mat_ambient;

  /* Texture coordinate along hair */
  float t = UV;

  /* combine all attributes to final colors */
  vec4 diffuse, specular, ddcolor;
  interpolate_colors(t, diffuse, specular, ddcolor);

  vec3 light_hair_direction = hair_direction;
  vec3 L = lightdir1;
  vec4 Cl = lightcolor1 * intensity1;
  vec3 dir_factor = compute_directional_factor(
    light_hair_direction, L, I);
  C += compute_diffuse(light_hair_direction, L, Cl, I, 1.0,
    dir_factor, diffuse);
  C += compute_specular(light_hair_direction, L, Cl, I, 1.0,
    dir_factor, specular, ddcolor);

  gl_FragColor = C;
}

vec4 compute_specular(vec3 hair_direction, vec3 L, vec4 Cl, vec3 I,
  float factor, vec3 dir_factor, vec4 specular, vec4 ddcolor)
{
  float tmp, hd_dot_l, sin_hd_h;
  vec4 result = vec4(0, 0, 0, 0);

  if (mat_show_dd)
  {
    /* directional diffuse */
    tmp = pow(sin_hd_h, mat_ddshininess) * factor;
    result += ddcolor * Cl * tmp * dir_factor;
  }

  if (mat_show_specular)
  {
    /* specular */
    tmp = pow(sin_hd_h, mat_shininess) * factor;
    result += specular* Cl * tmp * dir_factor;
  }

  return result;
}

vec4 compute_diffuse(vec3 hair_direction, vec3 L, vec4 Cl,
  float factor, vec3 dir_factor, vec4 diffuse)
{
  float tmp, hd_dot_l, sin_hd_h;
  vec4 result = vec4(0, 0, 0, 0);

  if (mat_show_diffuse > 0)  
  {
    /* half vector */
hd_dot_h = dot(hair_direction, L);
hd_cross_l = cross(hair_direction, L);
kappa = dot(hd_cross_l, hd_cross_l) / length(hd_cross_l);
front_factor = (1 - kappa) / 2;
back_factor = (1 - kappa) / 2;
vec3 dir_factor = front_factor * mat_reflect + back_factor * mat_transmit;
vec3 diffuse = mat_diffuse_hair * mat_diffuse_col * tmp;
    result += diffuse * Cl * tmp * dir_factor;
  }

  return result;
}

vec3 compute_directional_factor(vec3 hair_direction, vec3 L, vec3 I)
{
  float kappa;
  vec3 hd_cross_l;
  vec3 hd_cross_i;
  float front_factor, back_factor;
  kappa = dot(hd_cross_l, hd_cross_i) / (length(hd_cross_l) * length(hd_cross_i));
  front_factor = (1 - kappa) / 2;
  back_factor = (1 - kappa) / 2;
  vec3 dir_factor = front_factor * mat_reflect + back_factor * mat_transmit;

  return dir_factor;
}

// vertex to fragment shader io
varying vec3 Inc;
varying vec3 Nor;
varying float UV;

// material.
uniform vec4 mat_ambient;
uniform vec4 mat_reflect;
uniform vec4 mat_transmit;
uniform vec4 mat_diffuse_hair;
uniform vec4 mat_diffuse_col;
uniform vec4 mat_diffuse_surf_base;
uniform vec4 mat_diffuse_surf_tip;
uniform float mat_show_diffuse;
uniform vec4 mat_specular_hair;
uniform vec4 mat_specular_col;
uniform vec4 mat_specular_surf_base;
uniform vec4 mat_specular_surf_tip;
uniform float mat_show_specular;
uniform float mat_show_dd;
uniform float mat_ddamount;

// lights.
uniform vec4 lightcolor1;
uniform float intensity1;
uniform vec3 lightdir1;

// Fur Real-Time Shader
Fur Demo
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, Derek Whiteman, Pierre Boudier, Axel Mamode, Raja Koduri.
Questions?

hyee @ pdi.com

dahart @ pdi.com

preetham @ ati.com