SIGGRAPH 2002

SGI OpenGL Shader

Marc Olano
SGI
Interactive Rendering

Illusion of Presence

- 10 – 30 – 60 frames per second
- Immediate response
- Simple appearance
Multi-pass Rendering

Improved appearance

• Build effects
• Per-frame or per-object
• Still interactive

[Diefenbach97] [Peercy97] [Cabral99] [Kautz99]
What’s in a Pass?

Graphics hardware
- (as seen through OpenGL)

application

vertex operations  →  pixel operations

rasterize

fragment operations  →  frame buffer

texture
Rendering Passes

1. Application
2. Vertex Operations
3. Rasterize
4. Fragment Operations
5. Texture
6. Pixel Operations
7. Frame Buffer
Multi-Pass = SIMD

Single Instruction, Multiple Data

Classic SIMD

- Thousands/millions of processors
- Thinking Machines, PixelFlow, ...
- Not small-scale SIMD (MMX, etc.)

Shading languages use SIMD model

- Describe shading for one point
- Apply for every point on surface
Multi-Pass = SIMD

<table>
<thead>
<tr>
<th>General SIMD</th>
<th>OpenGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Control</td>
<td>Application</td>
</tr>
<tr>
<td>Processor Array</td>
<td>Pixel Array</td>
</tr>
<tr>
<td>Per-PE ALU</td>
<td>Fragment ops</td>
</tr>
<tr>
<td>Per-PE Memory</td>
<td>FB / Texture</td>
</tr>
<tr>
<td>Per-PE Conditionals</td>
<td>Alpha / Stencil</td>
</tr>
</tbody>
</table>
What’s it Mean?

We can create a compiler

- High-level language in
- OpenGL out
Isn’t That Slow?

No!

- Like drawing a few extra objects
- Optimize to compress passes
- Target hardware extensions
OpenGL Shader

ISL

OpenGL Shader

Plain OpenGL

IR extensions

NV extensions

Other extensions
About ISL

Things exposed in ISL

- Pass count: passes $\leq$ statements
  - Optimize to fewer
- Range: clamped $0 – 1$
- Texturing limits
  - No per-pixel computed texture coordinates
  - Can use per-vertex texture coordinates
Example 1

Shiny Bump Map

- Dependent texturing?
- ISL `lookup` function
Lookup-base Environment

ISL’s run anywhere philosophy

- 1D environment
- Internally, same as texture lookup
Can build with ISL code

```plaintext
repeat(h) {
    // Fresnel component
    uniform float f = fresnel(2*i/(n-1) - 1, refract);

    // color spline for ground
    groundsky[i] = spline(i/(h-1),{
        color(.3,.6,.1,f),
        color(.3,.6,.1,f),
        color(.6,.6,.1,f),
        color(.4,.7,.1,f),
        color(.4,.4,.1,f),
        color(.3,.3,.1,f})
    i = i+1;
}
```
Procedural Environment

And the sky...

```cpp
repeat(h) {
    // Fresnel component
    uniform float f = fresnel(2*i/(n-1) - 1, refract);

    // color spline for ground
    groundsky[i] = spline((i-h)/(h-1),{
        color(1.,1.,1.,f),
        color(1.,1.,1.,f),
        color(.3,.7,.9,f),
        color(.3,.7,.9,f),
        color(.3,.7,.9,f),
        color(.3,.7,.9,f)});
    i = i+1;
}
```
Bump Map

Makes smooth surface appear bumpy

Several choices

- Evaluate bump math per-fragment
- “Embossed” bumps
- Normal map
Normal Map

Normal range –1 to 1
Color range 0 to 1
Normal map is just a texture

- $R = 0.5 N_X + 0.5$
- $G = 0.5 N_Y + 0.5$
- $B = 0.5 N_Z + 0.5$
Bumped Normal Map

Perturb in tangent directions + renormalize
// rescale normal vectors from 0..1 to -1..1 and back
uniform matrix nScale = translate(-.5,-.5,-.5) *scale(2,2,2);
uniform matrix nUnscale = scale(.5,.5,.5) *translate(.5,.5,.5);

// transform -1..1 normal from object to world space
parameter matrix nm = inverse(affine(shadermatrix));

// set rgb to y (vertical) component and alpha to z
// so one lookup can do both environment map and Fresnel
uniform matrix ggggb = matrix(0, 0, 0, 0,
1, 1, 1, 0,
0, 0, 0, 1,
0, 0, 0, 0);
Actual shading code

```c
FB = texture(nmap);
FB = transform(nScale * nm * nUnscale * gggb);
FB = lookup(groundsky);
```
Demo
Example 2

Homomorphic BRDF Factorization
Factorization
Factored Textures

L
Shadowing

N
Microfacet

V
Masking
The Shader

```cpp
surface BRDF(
    uniform string brdfP = "brdf p.rgb";
    uniform string brdfQ = "brdf q.rgb";
    uniform color brdfC = color(1,1,1,1))
{
    FB = diffuse();
    FB *= texture(brdfP, 1, 1);
    FB *= texture(brdfQ, 1, 2);
    FB *= texture(brdfP, 1, 3);
    FB *= brdfC;
}
```
Texture Coordinates

texture(“texture”, matrix, \texttt{texcoord\_set})

Passed to application draw callback

- Per-vertex application code
- Vertex programs

OpenGL Shader 3.0

- includes vertex program emulation
Demo

Homomorphic BRDF Factorization
Example 3

Parameterized Wood
Need Some Bands

Start with a simple ramp

project("wave.bw",
    inverse(shadermatrix)*
    ringCenterXlate*
    ringAxisRotate*
    ringScale*
    textureCenterXlate);
if (FB[0] < lightToDark)
    FB = darkWood;
else
    FB = lightWood;
Differing Specular

FB = diffuse();
varying color dif=FB;

FB = environment("highlight.bw");
varying color spec=FB;

if (FB[0] < lightToDark) {
    FB = darkWood;
    FB *= dif;
    varying color a = FB;

    FB = darkGloss;
    FB *= spec;
    FB += a;
}


Turbulent Rings

FB = project("turbulence.bw", ...);
FB *= ringNoiseScale;
FB += project("wave.bw", ...);
FB = darkGloss;
FB.a = project("noise.bw", ...);
FB = over(darkGrainGloss);
FB *= spec;
FB += a;
Demo
Bonus Example

Level-of-detail Shaders
Level-of-detail Shaders

Add conditionals to adjust complexity

• Distance
• Importance
• Time
• Available texture
Level-of-detail

Automatic
- Add conditionals
- Change “hardware mapping” rules in each branch

Semi-automatic
- Use LOD building blocks

Manual
- Add conditionals
- Hand-code levels
Demo