Multi-Pass Shading

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

rasterize

fragment operations

pixel operations

texture

frame buffer
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 <= statements
  • Optimize to fewer

• Range: clamped 0 – 1

• Texturing limits
  • No per-pixel computed texture coordinates
  • Can use per-vertex texture coordinates
I Can do that Myself!

```c
surface fancy() {
    FB = environment("flowers.rgb");
    FB *= color(.5,.2,.0,0);
    FB = under(texture("birds.rgb");
    scale(2,2,2)));
}
```
#include <swizzle.h>
uniform color greentable[2] = 
{color(0,.2,0,1), color(0,.4,0,1)};
surface
toon( parameter float do_toon = 1.;
     parameter float edge = .25)
{
    FB = environment("redpark.env");
    if (do_toon > .5) {
        FB += edge;
        FB = transform(rgba_rrra);
        FB = lookup(greentable);
        FB += environment("sun.env");
    }
}
#include <swizzle.h>

uniform color greentable[2] =
{color(0,.2,0,1), color(0,.4,0,1)};

surface
toon( uniform float do_toon = 1.;
    uniform float edge = .25)
{
    FB = environment("redpark.env");
    if (do_toon > .5) {
        FB += edge;
        FB = transform(rgba_rrra);
        FB = lookup(greentable);
        FB += environment("sun.env");
    }
}
1: Simple Operations

\[
\begin{align*}
FB &= \text{environment}(\text{redpark}) \\
FB &= \text{environment}(\text{redpark}) \\
FB &= \text{environment}(\text{redpark}) \\
FB &= \text{environment}(\text{redpark}) \\
FB &= \text{environment}(\text{redpark}) \\
FB &= \text{environment}(\text{redpark}) \\
FB &= \text{environment}(\text{redpark}) \\
\end{align*}
\]

\[
\begin{align*}
a &= \text{texgen}(\text{environment}) \\
b &= \text{lookup}(\text{redpark}, a) \\
c &= \text{const}(0.25) \\
d &= \text{add}(b, c) \\
e &= \text{transform}(\text{rgba}_r\text{r}_r\text{r}_r\text{a}) \\
f &= \text{lookup}(\text{greentable}, e) \\
g &= \text{texgen}(\text{environment}) \\
h &= \text{lookup}(\text{sun}, g) \\
i &= \text{add}(f, h) \\
\end{align*}
\]
1: Map to Hardware

Pack passes back-to-front

Remove dead code

\[
\begin{align*}
a &= \text{texgen}(\text{environment}) \\
b &= \text{lookup}(\text{redpark}, a) \\
c &= \text{const}(0.25) \\
d &= \text{add}(b, c) \\
e &= \text{transform}(\text{rgba}_\text{rrra}, d) \\
f &= \text{lookup}(\text{greentable}, e) \\
g &= \text{texgen}(\text{environment}) \\
h &= \text{lookup}(\text{sun}, g) \\
i &= \text{add}(f, h)
\end{align*}
\]
1: Map to Hardware

Pack passes back-to-front

Remove dead code

\[ a = \text{texgen}(\text{environment}) \]
\[ b = \text{lookup}(\text{redpark}, a) \]
\[ c = \text{const}(0.25) \]
\[ d = \text{add}(b, c) \]
\[ e = \text{transform}(\text{rgba}_r\text{rra}, d) \]
\[ f = \text{lookup}(\text{greentable}, e) \]
\[ g = \text{texgen}(\text{environment}) \]
\[ h = \text{lookup}(\text{sun}, g) \]
\[ i = \text{add}(f, h) \]
1: Map to Hardware

Pack passes back-to-front

Remove dead code

\[ a = \text{texgen}(\text{environment}) \]
\[ b = \text{lookup}(\text{redpark},a) \]
\[ c = \text{const}.25 \]
\[ d = a + c \]
\[ e = \text{transform}(\text{rgba}_r, d) \]
\[ f = \text{lookup}(\text{greentable}, e) \]
\[ g = \text{texgen}(\text{environment}) \]
\[ h = \text{lookup}(\text{sun}, g) \]
\[ i = f + h \]
1: Map to Hardware

Pack passes back-to-front

Remove dead code

\[
\begin{align*}
    a &= \text{texgen}(\text{environment}) \\
    b &= \text{lookup}(\text{redpark}, a) \\
    c &= \text{const}(0.25) \\
    d &= \text{add}(b, c) \\
    e &= \text{transform}(\text{rgba}_r\text{rrr}_a, d) \\
    f &= \text{lookup}(\text{greentable}, e) \\
    g &= \text{texgen}(\text{environment}) \\
    h &= \text{lookup}(\text{sun}, g) \\
    i &= \text{add}(f, h)
\end{align*}
\]
1: Map to Hardware

Pack passes back-to-front

Remove dead code

\[ a = \text{texgen} \text{(environment)} \]
\[ b = \text{lookup} \text{(redpark,} a) \]
\[ c = \text{const} (.25) \]
\[ d = \text{add} \text{(} b, c) \]
\[ e = \text{transform} \text{(rgba}_\text{rrra}, d) \]
\[ f = \text{lookup} \text{(greentable,} e) \]
\[ g = \text{texgen} \text{(environment)} \]
\[ h = \text{lookup} \text{(sun,} g) \]
\[ i = \text{add} \text{(} f, h) \]
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
Wrap-up

Simple, easy-to-change description
• Flexible surface appearance
• Much easier than hand-coded GL

Portable

Enables powerful extra processing