Translating Shaders to Multi-Pass

Marc Olano
SGI

Sample RenderMan shader

```cpp
surface
beachball(

uniform float Ka = 1, Kd = 1;
uniform float Ks = .5, roughness = .1;
uniform color starcolor = color (1,.5,0);
uniform color bandcolor = color (1,.2,.2);
uniform float rmin = .15, rmax = .4;
uniform float npoints = 5;
)
{

color Ct;
float angle, r, a, in_out;
uniform float starangle = 2*PI/npoints;
uniform point p0 = rmax*point(cos(0),sin(0),0);
uniform point p1 = rmin*point(cos(starangle/2),sin(starangle/2),0);
uniform vector d0 = p1 - p0;
vector d1;

angle = 2*PI * s;
r = .5-abs(t-.5);
a = mod(angle, starangle)/starangle;

if (a >= 0.5)
    a = 1 - a;

d1 = r*(cos(a), sin(a),0) - p0;
in_out = step(0, zcomp(d0^d1));
Ct = mix(mix(Cs, starcolor, in_out), bandcolor, step(rmax,r));

/* specular shading model */
normal Nf = normalize(faceforward(N,I));
Oi = Os;
Ci = Os * (Ct * (Ka * ambient() +
            Kd * diffuse(Nf)) +
            Ks * specular(Nf,-normalize(I),roughness));
}
```

Passes for varying computation

Each line of text below is a pass used as part of the varying computation in the above shader. Corresponding images appear to the left, though no image appears for any pass that does not change the framebuffer. No optimizations are included in this example as they can make the correspondence between source code and passes harder to follow.

```cpp
// set stencil for masking in later passes

angle = 2*PI * s;                       // draw geometry with ‘s’ as color
```
\[\text{angle} = 2\pi \times s;\] // use blend to multiply by 2*PI
\[\text{angle} = 2\pi \times s;\] // store in texture named "angle"

\[\text{r} = 0.5 - \text{abs}(t - 0.5);\] // draw geometry with 't' as color
\[\text{r} = 0.5 - \text{abs}(t - 0.5);\] // use blend to subtract .5
\[\text{r} = 0.5 - \text{abs}(t - 0.5);\] // copy through "abs" color table
\[\text{r} = 0.5 - \text{abs}(t - 0.5);\] // blend: subtract from .5
\[\text{r} = 0.5 - \text{abs}(t - 0.5);\] // store in texture named "r"

\[\text{a} = \text{mod(angle, starangle)}/\text{starangle};\] // load "angle" from texture
\[\text{a} = \text{mod(angle, starangle)}/\text{starangle};\] // blend: multiply by 1/starangle
\[\text{a} = \text{mod(angle, starangle)}/\text{starangle};\] // copy through "floor" color table
\[\text{a} = \text{mod(angle, starangle)}/\text{starangle};\] // blend: multiply by starangle
\[\text{a} = \text{mod(angle, starangle)}/\text{starangle};\] // blend: subtract from "angle"
\[\text{a} = \text{mod(angle, starangle)}/\text{starangle};\] // blend: multiply by 1/starangle
\[\text{a} = \text{mod(angle, starangle)}/\text{starangle};\] // blend: multiply by 1/starangle
\[\text{a} = \text{mod(angle, starangle)}/\text{starangle};\] // store in texture named "a"
if (\text{a} >= 0.5) // load "a" from texture

if (\text{a} >= 0.5) // blend: subtract .5

if (\text{a} >= 0.5) // alpha test: set stencil mask

\[\text{a} = 1 - \text{a};\] // load "a" from texture

\[\text{a} = 1 - \text{a};\] // blend: subtract from 1
\[
a = 1 - a; \quad \text{// load "a" & combine with stencil}
\]
\[
a = 1 - a; \quad \text{// store in texture named "a"}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// load "a" from texture}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// copy through "cos" color table}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// store in texture named "ftemp0"}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// load "a" from texture}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// copy through "cos" color table}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// store in texture named "ftemp1"}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// load constant value of 0}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// load "ftemp0" into red}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// load "ftemp1" into green}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// blend: multiply by texture "r"}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// blend: subtract uniform p0}
\]
\[
d_1 = r \cdot (\cos(a), \sin(a), 0) - p_0; \quad \text{// store in texture named "d1"}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// load uniform d0}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// color matrix: store in yzx order in "ctemp0"}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// color matrix: store in zxy order in "ctemp1"}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// load "d1" from texture}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// color matrix: store in yzx order in "ctemp2"}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// color matrix: shuffle to zxy order}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// blend: multiply by "ctemp0"}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// store back into "ctemp0"}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// load "ctemp1" from texture}
\]
\[
in_{\text{out}} = \text{step}(0, z\text{comp}(d_0 \cdot d_1)); \quad \text{// blend: multiply by "ctemp2"}
in_out = step(0, zcomp(d0^d1)); // blend: subtract "ctemp0"

in_out = step(0, zcomp(d0^d1)); // blend: subtract "ctemp0"

in_out = step(0, zcomp(d0^d1)); // color matrix: copy z to all channels
in_out = step(0, zcomp(d0^d1)); // blend: subtract 0 (to shift step)

in_out = step(0, zcomp(d0^d1)); // copy through "step" color table
in_out = step(0, zcomp(d0^d1)); // store in texture named "in_out"

...mix(Cs, starcolor, in_out)... // load uniform Cs
...mix(Cs, starcolor, in_out)... // load "in_out" into alpha

...mix(Cs, starcolor, in_out)... // blend: mix Cs and starcolor
// store in texture named "ctemp0"

...mix(..., bandcolor, step(rmax,r)); // load "r" from texture

...mix(..., bandcolor, step(rmax,r)); // blend: subtract from rmax

...mix(..., bandcolor, step(rmax,r)); // copy through "step" color table
// store in texture named "ftemp0"

...mix(..., bandcolor, step(rmax,r)); // load "ctemp0"
...mix(..., bandcolor, step(rmax,r)); // load "ftemp0" into alpha

...mix(..., bandcolor, step(rmax,r)); // blend: mix with bandcolor
Ct = mix(...); // store in texture named "Ct"

...normalize(faceforward(N,I)); // draw geometry, with 'I' as color
// store in texture named "ctemp0"

...normalize(faceforward(N,I)); // draw geometry, with 'Ng' as color

...normalize(faceforward(N,I)); // blend: multiply by texture "ctemp0"

...normalize(faceforward(N,I)); // color matrix: add x+y+z

...normalize(faceforward(N,I)); // copy through "flip" color table
...normalize(faceforward(N,I)); // blend: draw 'N' & multiply
normal Nf = normalize(...); // blend: multiply (to square)
normal Nf = normalize(...); // color matrix: sum channels
normal Nf = normalize(...); // copy through "invsqrt" color table
normal Nf = normalize(...); normal Nf = normalize(...); // store in texture named "Nf"

Ci = ...(... + Kd * diffuse(Nf))... // Lighting passes omitted
Ci = ...Ks * specular(...)...; // Lighting passes omitted
Ci = ...Ct * (...)... // blend: multiply by "Ct"
Ci = Os * (...); // blend: add "ctemp0" (diffuse & ambient)
Ci = Os * (...); // load "Ci" outside object using stencil
Ci = Os * (...); // store combined Ci into texture named "Ci"