Translating Shaders to Multi-Pass

Marc Olano
SGI

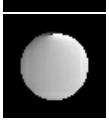
Sample RenderMan shader

```
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.

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



```
angle = 2*PI * s;                    // draw geometry with 's' as color
```

```
angle = 2*PI * s;                    // use blend to multiply by 2*PI
angle = 2*PI * s;                    // store in texture named "angle"
```

```
r = .5-abs(t-.5);                    // draw geometry with 't' as color
```

```
r = .5-abs(t-.5);                    // use blend to subtract .5
```

```
r = .5-abs(t-.5);                    // copy through "abs" color table
```

```
r = .5-abs(t-.5);                    // blend: subtract from .5
r = .5-abs(t-.5);                    // store in texture named "r"
```

```
a = mod(angle, starangle)/starangle; // load "angle" from texture
```

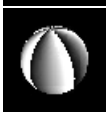
```
a = mod(angle, starangle)/starangle; // blend: multiply by 1/starangle
```

```
a = mod(angle, starangle)/starangle; // copy through "floor" color table
```

```
a = mod(angle, starangle)/starangle; // blend: multiply by starangle
```

```
a = mod(angle, starangle)/starangle; // blend: subtract from "angle"
```

```
a = mod(angle, starangle)/starangle; // blend: multiply by 1/starangle
```

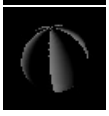


```
a = mod(angle, starangle)/starangle; // blend: multiply by 1/starangle
a = mod(angle, starangle)/starangle; // store in texture named "a"
if (a >= 0.5)                        // load "a" from texture
```

```
if (a >= 0.5)                        // blend: subtract .5
```

```
if (a >= 0.5)                        // alpha test: set stencil mask
```

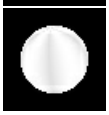
```
    a = 1 - a;                       // load "a" from texture
```



```
    a = 1 - a;                       // blend: subtract from 1
```

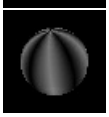
```
    a = 1 - a;                      // load "a" & combine with stencil
    a = 1 - a;                      // store in texture named "a"
```

```
d1 = r*(cos(a), sin(a),0) - p0;     // load "a" from texture
```



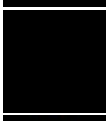
```
d1 = r*(cos(a), sin(a),0) - p0;     // copy through "cos" color table
                                    // store in texture named "ftemp0"
```

```
d1 = r*(cos(a), sin(a),0) - p0;     // load "a" from texture
```



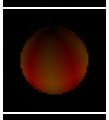
```
d1 = r*(cos(a), sin(a),0) - p0;     // copy through "cos" color table
                                    // store in texture named "ftemp1"
```

```
d1 = r*(cos(a), sin(a),0) - p0;     // load constant value of 0
```

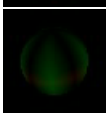


```
d1 = r*(cos(a), sin(a),0) - p0;     // load "ftemp0" into red
```

```
d1 = r*(cos(a), sin(a),0) - p0;     // load "ftemp1" into green
```

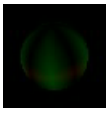


```
d1 = r*(cos(a), sin(a),0) - p0;     // blend: multiply by texture "r"
```

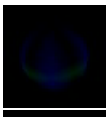


```
d1 = r*(cos(a), sin(a),0) - p0;     // blend: subtract uniform p0
d1 = r*(cos(a), sin(a),0) - p0;     // store in texture named "d1"
```

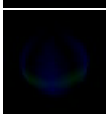
```
in_out = step(0, zcomp(d0^d1));     // load uniform d0
                // color matrix: store in yzx order in "ctemp0"
                // color matrix: store in zxy order in "ctemp1"
```



```
in_out = step(0, zcomp(d0^d1));     // load "d1" from texture
                // color matrix: store in yzx order in "ctemp2"
```

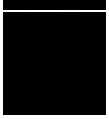


```
in_out = step(0, zcomp(d0^d1));     // color matrix: shuffle to zxy order
```

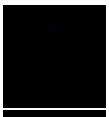


```
in_out = step(0, zcomp(d0^d1));     // blend: multiply by "ctemp0"
                                    // store back into "ctemp0"
```

```
in_out = step(0, zcomp(d0^d1));     // load "ctemp1" from texture
```

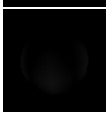


```
in_out = step(0, zcomp(d0^d1));     // blend: multiply by "ctemp2"
```



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

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



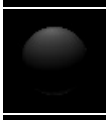
```
in_out = step(0, zcomp(d0^d1));      // color matrix: copy z to all channels
in_out = step(0, zcomp(d0^d1));      // blend: subtract 0 (to shift step)
```

```
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"
```

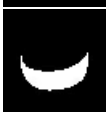
```
...mix(Cs, starcolor, in_out)...     // load uniform Cs
...mix(Cs, starcolor, in_out)...     // load "in_out" into alpha
```

```
...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
```

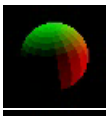


```
...mix(..., bandcolor, step(rmax,r));// copy through "step" color table
                                     // store in texture named "ftemp0"
```

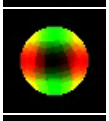
```
...mix(..., bandcolor, step(rmax,r));// load "ctemp0"
...mix(..., bandcolor, step(rmax,r));// load "ftemp0" into alpha
```

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

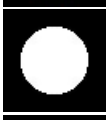
```
...normalize(faceforward(N,I));      // draw geometry, with 'I' as color
                                     // store in texture named "ctemp0"
```



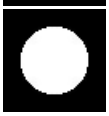
```
...normalize(faceforward(N,I));      // draw geometry, with 'Ng' as color
```



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

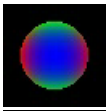


```
...normalize(faceforward(N,I));      // color matrix: add x+y+z
```

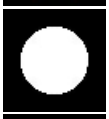


```
...normalize(faceforward(N,I));      // copy through "flip" color table
```

```
...normalize(faceforward(N,I));      // blend: draw 'N' & multiply
                                     // store in texture named "ctemp0"
```

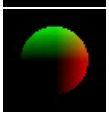


```
normal Nf = normalize(...);          // blend: multiply (to square)
```

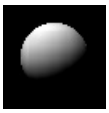


```
normal Nf = normalize(...);          // color matrix: sum channels
```

```
normal Nf = normalize(...);          // copy through "invsqrt" color table
```

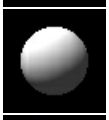


```
normal Nf = normalize(...);          // blend: multiply by texture "ctemp0"
normal Nf = normalize(...);          // store in texture named "Nf"
```

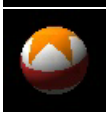


```
Ci = ...(... + Kd * diffuse(Nf))...  // Lighting passes omitted
                                     // store in texture named "ctemp0"
```

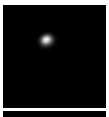
```
Ci = ...(Ka * ambient() + ...)...    // Lighting passes omitted
```



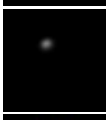
```
Ci = ...(Ka * ambient() + ...)...    // blend: add "ctemp0"
```



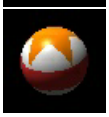
```
Ci = ...Ct * (...)...                // blend: multiply by "Ct"
                                     // store in texture named "ctemp0"
```



```
Ci = ...Ks * specular(...));         // Lighting passes omitted
```



```
Ci = ...Ks * specular(...));         // blend: multiply by uniform 'Ks'
```



```
Ci = Os * (...);                     // blend: add "ctemp0" (diffuse & ambient)
Ci = Os * (...);                     // blend: multiply by Os
Ci = Os * (...);                     // load "Ci" outside object using stencil
Ci = Os * (...);                     // store combined Ci into texture named "Ci"
```