An Introduction to RenderMan Shaders for all you GLSLers!

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

Why Are We Even Talking About This?

1. You never know – in the future you might be in a position to use RenderMan in the making of movies, TV commercials, etc.
2. You can get RenderMan for free for non-commercial use. It is fun to experiment with.
3. You will be surprised how close what you know now matches what you need to know to run RenderMan shaders. (Congratulations!)

You can get the non-commercial version of RenderMan by starting here:

https://renderman.pixar.com/intro

History of RenderMan, I


1979: Ed Catmull, Alvy Ray Smith, and others leave NYIT to form the Computer Division of Lucasfilm

1984: John Lasseter leaves Disney Animation to join Pixar

History of RenderMan, II

Pixar Image

Computer Rendering

Software

REYES

Star Trek II (1982)

Digital Editing and Compositing

Image/Volume Rendering Hardware

Pixar Animation Studios

RIB

Shade Trees

prman

History of RenderMan, III

Pixar Animation Studios

1986: Luxo Jr.: Nominated for an Academy Award

1988: Tin Toy – won Academy Award for Best Animated Short

1993: RenderMan wins a Technical Academy Award

1995: Toy Story

Early Uses of RenderMan in Movies
### RenderMan Software Rendering Pipeline

1. RIB File
2. Bounding Box Analysis
3. Split
4. Dice into Microfacets
5. Call the Shaders
6. Do Front-to-Back Compositing
7. Assemble the Pixels
8. Final Image

### RenderMan Composites Starting at the Eye

First, let’s think about it back-to-front:

\[
\text{color}_{i} = \alpha_i \text{color}_i + (1 - \alpha_i) \text{black},
\]

\[
\text{color}_{i+1} = \alpha_i \text{color}_{i+1} + (1 - \alpha_i) \text{color}_{i+2},
\]

\[
\text{color}^* = \alpha_i \text{color}_i + (1 - \alpha_i) \text{color}_i.
\]

Substituting gives us the front-to-back equation:

\[
\text{color}^* = \alpha_i \text{color}_i + (1 - \alpha_i) \alpha_i \text{color}_i + (1 - \alpha_i) (1 - \alpha_i) \text{color}_i + (1 - \alpha_i) (1 - \alpha_i) \text{black}.
\]

### RenderMan Renders at Higher-than-Screen-Resolution

### All Six RenderMan Shader Types

1. Displacement
2. Distortion / transformation
3. Surface
4. Lighting
5. Atmospheric / volumetric
6. Imaging

RenderMan **Built-in Microfaceting** Manual or GLSL Tessellation
### Fundamental Differences Between RenderMan Shaders and GLSL Shaders

<table>
<thead>
<tr>
<th>Topic</th>
<th>RenderMan</th>
<th>GLSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>1. Speed, 2. Image quality</td>
<td>1. Speed, 2. Image quality</td>
</tr>
<tr>
<td>Shader Types</td>
<td>Surface, Displacement (+4 others)</td>
<td>Vertex, Fragment, Geometry, Tessellation, Compute</td>
</tr>
<tr>
<td>Surface Preprocessing</td>
<td>Microfaces</td>
<td>None (± Tessellation shaders)</td>
</tr>
<tr>
<td>Recompute Normals</td>
<td>Calculate/normal</td>
<td>None (± Tessellation shaders)</td>
</tr>
<tr>
<td>Getting Rid of Phase</td>
<td>Discard;</td>
<td></td>
</tr>
<tr>
<td>Shader Variables</td>
<td>Uniform, Varying</td>
<td></td>
</tr>
<tr>
<td>Coordinate Systems</td>
<td>Shader (+Model), World</td>
<td>Model (+VGC), Eye (HRC)</td>
</tr>
<tr>
<td>Noise</td>
<td>Built-in</td>
<td></td>
</tr>
<tr>
<td>Complex Shaders</td>
<td>Must do yourself</td>
<td>Driver does it for you</td>
</tr>
<tr>
<td>Complex messages</td>
<td>Crypto</td>
<td>Crypto</td>
</tr>
</tbody>
</table>

### RIB Commands, I

```
Display "outputimage.tif" "srf" "rgb"
Display "outputimage.shd" "shadow" "z"
Imager "background" "color" [r g b]
Clipping "znear" "zfar"
Format 1280 1024 1.0
PixelSamples 2 2
PixelFilter "Gaussian" 4 4
Projection "perspective" "fov" 60.
ScreenWindow "left" "right" "bottom" "top"
Translate tx ty tz
Rotate deg ix iy iz
Scale sx sy sz
WorldBegin
WorldEnd
AttributeBegin
AttributeEnd
Color [ r g b ]
Opacity [ r g b ]
Surface "Plastic" "Ka" ka "Kd" kd "Ks" ks "roughness" r "specularcolor" [r g b]
LightSource "ambientlight" num "intensity" i "lightcolor" [r g b]
LightSource "distantlight" num "intensity" i "lightcolor" [r g b] "from" [x y z] "to" [x y z]
```

### RIB Commands, II

```
Polygon "P" [x0 y0 z0 x1 y1 z1 ...] "at" [x0 y0 z1 x1 y1 z1 ...]
Points "P" [x0 y0 z0 x1 y1 z1 ...] "constantwidth" w
Points "P" [x0 y0 z0 x1 y1 z1 ...] "width" [w0 w1 ...]
Sphere radius zmin zmax sweepAngle
Cylinder radius zmin zmax sweepAngle
Cone height radius sweepDegrees
Torus majorRadius minorRadius startAngle endAngle sweepMagnitude
Hyperboloid x0 y0 z0 x1 y1 z1 sweepAngle
Paraboloid zmaxRadius zmin zmax sweepAngle
Disk xheight radius sweepAngle
Basis "Bezier" [4] "nonperiodic" "P" [x0 y0 z0 x1 y1 z1 ...] "constantwidth" [w]
```

### Cartesian (X) Stripes

#### stripes.rib

```
stripes.rib

RIB/RenderMan RIB
version 3.03
Declare "Prob" "uniform float"
Display "stripes.tif" "srf" "rgb"
Format 512 512 -1
ShadingRate 1
LightSource "ambientlight" 1 "intensity" [0.25]
LightSource "distantlight" 2 "intensity" [0.75] "from" [0 0 -10] "to" [0 0 0]
Projection "perspective" "fov" [70]
WorldBegin
Translate 0 0 8
Surface
stripes
Ci = 0;
Color [1 1 1]
Opacity [1 1 1]
Sphere 3 -3 3 360
Translate 2. 0. 0.
WorldEnd
```

#### Cartesian (X) Stripes

```
Cartesian (X) Stripes

#### stripes.slo

```
Cartesian (X) Stripes

```
```
## RenderMan RIB

version 3.03

Declare "Prob" "uniform float"

Display "rings.tiff" "file" "rgb"
Format 500 500 -1
ShadingRate 1

LightSource "ambientlight" 1 "intensity" [0.25]
LightSource "distantlight" 2 "intensity" [0.75] "from" [5 0 -10] "to" [0 0 0]

Projection "perspective" "fov" [70]

WorldBegin

Translate 0 0 8

Surface "rings" "Prob" 0.6
Color [1 1 1]
Opacity [1 1 1]

Sphere 3 -3 3 360
Translate 2. 0. 0.

WorldEnd

## Rings (= Polar Stripes)

surface rings { float
Prob = 0.4,
Ka = 0.5,
Kd = 0.5,
Ks = 1.,
roughness = 0.1;
color specularcolor = color( 1, 1, 1 )
}

varying vector Nf = faceforward( normalize( N ), I );

float x = xcomp( P );
float y = ycomp( P );
float r = sqrt( x*x + y*y );
float rfrac = mod( r, 1. );

if( rfrac < Prob )

Ci = color( 1., .5, 0. );
else

Ci = Cs;

Oi = 1.;

Ci = Oi * ( Ci * ( Ka * ambient( ) + Kd * diffuse(Nf) ) +
specularcolor * Ks * specular( Nf, V, roughness ) );

}

## Dots

Display "dots.tiff" "file" "rgb"
Format 512 512 -1
ShadingRate 1

LightSource "ambientlight" 1 "intensity" [0.25]
LightSource "distantlight" 2 "intensity" [0.75] "from" [5 0 -10] "to" [0 0 0]

Projection "perspective" "fov" [70]

WorldBegin

Translate 0 0 6

Surface "dots" "Diam" 0.10
Color [1 1 1]
Opacity [1 1 1]

TransformBegin

Rotate 90 1. 0. 0.

Sphere 3 -3 3 360

TransformEnd

WorldEnd

## Dots

surface dots { float
Diam = 0.10, // dot diameter
Ks = 0.5,
Kd = 0.5,
Ka = .1,
roughness = 0.1;
color specularColor = color( 1, 1, 1 )
}

float up = 2. * u;
float vp = v;
float numinu = floor( up / Diam );
float numinv = floor( vp / Diam );

color dotColor = Cs;

if( mod( numinu+numinv, 2 ) == 0 )

{
float uc = numinu*Diam + Diam/2.;
float vc = numinv*Diam + Diam/2.;
up = up - uc;
vp = vp - vc;
point upvp =  point( up, vp, 0. );
point cntr =  point( 0., 0., 0. );
if( distance( upvp, cntr ) < Diam/2. )

{ dotColor = color( 1., .5, 0. ); // beaver orange?
}
}

varying vector Nf = faceforward( normalize( N ), I );

vector V = normalize( -I );
Oi = 1.;

Ci = Oi * ( dotColor * ( Ka * ambient( ) + Kd * diffuse(Nf) ) +
specularColor * Ks * specular( Nf, V, roughness ) );
Hallmark has even made a Christmas tree ornament out of it!