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

1977: Star Wars IV: A New Hope

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

Image Processing
Digital Editing and Compositing
Effects
Image/Volume Rendering Hardware

1984: John Lassiter leaves Disney Animation to join Pixar
History of RenderMan, II

- Pixar Image Computer
- Image/Volume Rendering Hardware
- Rendering Software
- REYES
- RenderMan
- RIB
- Shade Trees
- prman
- Pixar Animation Studios

- Star Trek II (1982)
- Young Sherlock Holmes (1985)
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
- Up, 2009
- Toy Story 2, 1999
- Toy Story 3, 2010
- Toy Story 4, 2019
- A Bug’s Life, 1998
- A Bug’s Life, 2001
- Finding Nemo, 2003
- Finding Dory, 2016
- Coco, 2017
- Brave, 2012
- The Incredibles, 2004
- The Incredibles, 2016
- Wall-E, 2008
- Monsters University, 2013
- Ratatouille, 2007
- Cars, 2006
- Cars 2, 2011
- Inside Out, 2015
- Monsters University, 2013
- The Good Dinosaur, 2015
- Soul, 2021
Tangled, 2010
Moana, 2016
Frozen, 2013
Frozen II, 2019
RIB File

Bounding Box Analysis

Split

Dice into Microfacets

Call the Shaders

Do Front-to-Back Compositing

Assemble the Pixels

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

\[
\text{color}_{12} = \alpha_2 \text{color}_2 + (1 - \alpha_2) \text{black},
\]

\[
\text{color}_{01} = \alpha_1 \text{color}_1 + (1 - \alpha_1) \text{color}_{12},
\]

\[
\text{color}^* = \alpha_0 \text{color}_0 + (1 - \alpha_0) \text{color}_{01}.
\]

Substituting gives us the front-to-back equation:

\[
\text{color}^* = \alpha_0 \text{color}_0 + (1 - \alpha_0) \alpha_1 \text{color}_1 + (1 - \alpha_0)(1 - \alpha_1) \alpha_2 \text{color}_2 + (1 - \alpha_0)(1 - \alpha_1)(1 - \alpha_2) \text{black}.
\]
RenderMan Renders at Higher-than-Screen-Resolution
All Six RenderMan Shader Types

1. Displacement  $\approx$ GLSL Vertex Shader
2. Distortion / transformation
3. Surface  $\approx$ GLSL Fragment Shader
4. Lighting
5. Atmospheric / volumetric
6. Imaging

RenderMan *Built-in Microfaceting*  $\approx$ 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><strong>Goals</strong></td>
<td>1. Image quality, 2. Speed</td>
<td>1. Speed, 2. Image quality</td>
</tr>
<tr>
<td><strong>Shader Types</strong></td>
<td>Surface, Displacement (+4 others)</td>
<td>Vertex, Fragment, Geometry, Tessellation, Compute</td>
</tr>
<tr>
<td><strong>Surface Preprocessing</strong></td>
<td>Microfacets</td>
<td>None [ ± Tessellation shaders]</td>
</tr>
<tr>
<td><strong>Recompute Normals</strong></td>
<td>CalculateNormal</td>
<td>None</td>
</tr>
<tr>
<td><strong>Getting Rid of Pixels</strong></td>
<td>Oi = 0.;</td>
<td>discard;</td>
</tr>
<tr>
<td><strong>Surface/Fragment shader sets</strong></td>
<td>R, G, B, αr, αg, αb</td>
<td>R, G, B, A [,Z]</td>
</tr>
<tr>
<td><strong>Shader Variables</strong></td>
<td>Uniform, Varying</td>
<td>Attribute, Uniform, Out, In</td>
</tr>
<tr>
<td><strong>Coordinate Systems</strong></td>
<td>Shader (=Model), World</td>
<td>Model (=OC), Eye (≈WC)</td>
</tr>
<tr>
<td><strong>Noise</strong></td>
<td>Built-in</td>
<td>Somewhat built-in or use a Texture</td>
</tr>
<tr>
<td><strong>Compile Shaders</strong></td>
<td>Must do yourself</td>
<td>Driver does it for you</td>
</tr>
<tr>
<td><strong>Compiler messages</strong></td>
<td>Cryptic</td>
<td>Cryptic</td>
</tr>
</tbody>
</table>
RIB Commands, I

Display "outputimage.tiff" "tiff" "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 xleft xright ybottom ytop
Translate tx ty tz
Rotate deg rx ry rz
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" l "lightcolor" [r g b] "from" [x y z] "to" [x y z]
### RIB Commands, II

<table>
<thead>
<tr>
<th>Object</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polygon</td>
<td>“P” [x0 y0 z0  x1 y1 z1 …] “st” [s0 t0  s1 t1 …]</td>
</tr>
<tr>
<td>Points</td>
<td>“P” [x0 y0 z0  x1 y1 z1 …] “constantwidth” w</td>
</tr>
<tr>
<td>Points</td>
<td>“P” [x0 y0 z0  x1 y1 z1 …] “width” [w0 w1 …]</td>
</tr>
<tr>
<td>Sphere</td>
<td>radius zmin zmax sweepAngle</td>
</tr>
<tr>
<td>Cylinder</td>
<td>radius zmin zmax sweepAngle</td>
</tr>
<tr>
<td>Cone</td>
<td>height radius sweepDegrees</td>
</tr>
<tr>
<td>Torus</td>
<td>majorRadius minorRadius startAngle endAngle sweepAngle</td>
</tr>
<tr>
<td>Hyperboloid</td>
<td>x0 y0 z0  x1 y1 z1 sweep_Angle</td>
</tr>
<tr>
<td>Paraboloid</td>
<td>zmaxRadius zmin zmax sweepAngle</td>
</tr>
<tr>
<td>Disk</td>
<td>xHeight radius sweepAngle</td>
</tr>
<tr>
<td>Basis</td>
<td>“Bezier” 3</td>
</tr>
<tr>
<td>Curves</td>
<td>“cubic” [4] “nonperiodic” “P” [x0 y0 z0  x1 y1 z1 …] “constantwidth” [w]</td>
</tr>
</tbody>
</table>

TransformEnd
TransformBegin
Cartesian (X) Stripes

stripes.rib

```rib
##RenderMan RIB
version 3.03
Declare "Prob" "uniform float"

Display "stripes.tiff" "file" "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" "Prob" 0.8
  Color [1 1 1]
  Opacity [1 1 1]
  Sphere 3 -3 3 360
  Translate 2. 0. 0.
WorldEnd
```

Tells RenderMan to start using a surface shader file called `stripes.slo`

Says that we want to set the `Prob` argument in that shader to **0.8**
surface
stripes( float
    Prob = 0.4, // probability of seeing orange
    Ks = 0.5,
    Kd = 0.5,
    Ka = .1,
    roughness = 0.1;
    color specularColor = color( 1, 1, 1 )
)
{
    color stripeColor;

    float x = xcomp( P );
    //x = x + 0.30 * sin( 1. * ycomp(P) ); // adding a wobble
    float xfrac = mod( x, 1. );

    if( xfrac < Prob )
        stripeColor = color( 1., .5, 0. ); // beaver orange?
    else
        stripeColor = Cs;

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

    Oi = 1.;
    Ci = Oi * ( stripeColor * ( Ka * ambient( ) + Kd * diffuse(Nf) ) +
                specularColor * Ks * specular( Nf, V, roughness ) );}
Cartesian (X) Stripes
Rings

##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
surface
rings( float
    Prob = 0.4,
    Ks = 0.5,
    Kd = 0.5,
    Ka = .1,
    roughness = 0.1;
    color specularcolor = color( 1, 1, 1 )
)
{
    varying vector Nf = faceforward( normalize( N ), I );
    vector V = normalize( -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 ) );
Rings (= Polar Stripes)
##RenderMan RIB version 3.03
Declare "Diam" "uniform float"

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 8 -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
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 ) ) );
Dots
Hallmark has even made a Christmas tree ornament out of it!