Introduction to RenderMan

Saty Raghavachary
Talk outline

RenderMan basics

- origins
- 'RI Spec' – brief tour
- workflow
- scene (RIB) files
- shaders (RSL)
- resources
Origins, history

- Univ. of Utah – Ed Catmull
- Lucasfilm, Pixar - “REYES” (’87)
- Photorealistic RenderMan (’89), RSL (’90)
Origins, history

REYES

- all objects get broken down into micropolygons
- micropolygons are shaded
- results are combined to produce output pixel colors and opacities
Origins, history

Pixar’s short films
- Luxo Jr. (1986)
- Red’s Dream (1987)
- Tin Toy (1988)
- Knickknack (1989)
- Geri’s Game (1997)
- For the Birds (2000)
- Mike’s New Car (2002)
- Boundin’ (2004)
Origins, history
Origins, history

LOTS of movies use RenderMan!
What is RenderMan?

- an interface between modelers and renderers

- idea is to be able to mix-and-match them
RI Spec

What's in the Spec

- core capabilities (eg. hierarchical graphics state, antialiasing, programmable shading language)
- advanced/optional capabilities (eg. motion blur, depth of field, global illumination)
- C/C++ bindings and RenderMan Interface Bytestream (RIB) bindings
- RenderMan Shading Language (RSL)
RI Spec

Who implements the Spec?

- Pixar – Photorealistic RenderMan (PRMan) - v.13
- Dot C Software – RenderDotC
- Paul Gregory and team - Aqsis
- SiTex Graphics – AIR
- Okan Arikan – Pixie
- 3Delight
- ... several others ...
RI Spec

Version numbers

• 3.0 – 1988
• 3.1 – 1989, revised in 1995
• 3.2 – 2000
• 3.2.1 – Nov. 2005
• 3.3 – in the works

Read the Spec thoroughly!
Workflow

RenderMan dataflow/pipeline

Do modeling, layout, animation, effects and lighting/surfacing elsewhere (eg. in Maya)
Workflow

Sources of RIB files

- 'by hand', using a text editor
- by running a program – bindings exist for Java, Python, Perl, Tcl, Ruby..
- standalone scene converters, eg. mi2rib
- native output, eg. from Blender
- translator plugins – eg. for Maya, we have MTOR/Slim, RfM, MayaMan and Liquid
Sources of shaders

- from scratch, 'by hand'
- plugin interface, eg. 'Slim' for Maya
- integrated into host program, eg. 'RenderMan for Maya' plugin for Maya
- standalone 'connect the boxes' UI programs, eg.:
  - ShaderMan
  - ShadeTree (defunct)
Workflow

ShaderMan, ShadeTree
Workflow

Sources of maps
• digital paintings, scans, photos..
• prior renders, ie. from a previous 'pass'

Types of maps:
– texture
– environment/reflection
– normal
– shadow/deep shadow
– photon
– irradiance cache
– occlusion
– brick

TIFF images for texturing need to be converted to map format, eg. .tex for PRMan
Workflow

Ways of extending RenderMan

• shaders
• DSO shader plugins, “shadeops”
• standalones/DSOs to output RIB for use with the 'Procedural' RIB call
• display driver DSOs
RIB files

- scene description files
- simple, declarative style, eg. Projection “perspective” “fov” 54.3
- no loops, branches, function calls etc.
- can be ASCII or binary
- usually one file per frame of animation
RIB files

A very simple RIB file and output:

```
# RIB file to render a sphere on a plane
version 3.03
Display "pln_sph.tiff" "file" "rgba"
Format 600 400 1
Projection "perspective" "fov" 30
Translate 0 0 15
Rotate -15 1 0 0
Rotate -20 0 1 0
WorldBegin
LightSource "distantlight" 2 "intensity" .5
   "from" [-20 -20 -20] "to" [0 0 0]
AttributeBegin
Surface "matte"
Color [1 0 0]
Translate 0 1 0
Sphere 1 -1 1 360
AttributeEnd
AttributeBegin
Color [0 0 1]
Surface "matte"
Scale 5 5 5
Patch "bilinear" "P" [-.5 -.5 .5 .5
   -.5 .5 -.5 .5]
AttributeEnd
WorldEnd
```

PRMan render command: `render sph_pln.rib`
Language highlights

• C-like syntax
• types: float, string, color, point, normal...
• operators: +, -, *, /, %, ==...
• control statements: if(), for()...
• rich collection of built-in functions
• user-definable functions
• DSO shadeops (shader plugins)
• pre-processor directives, eg. #include
Built-in function categories

- mathematical, eg. `asin()`
- geometric, eg. `normalize()`
- color, eg. `ctransform()`
- matrix, array, eg. `arraylength()`, `determinant()`
- string, eg. `concat()`
- shading, lighting, eg. `diffuse()`, `trace()`
- texture mapping, eg. `texture()`, `shadow()`
- message passing, info., eg. `surface()`, `rayinfo()`
Five categories of shaders

- surface
- displacement
- atmosphere
- light
- imager

Shader source needs to be compiled, eg. (PRMan):
\texttt{shader myshader.sl}
Surface shader global variables

In addition:
- Ci, Oi [for output]
- s, t and u, v
- dPdu, dPdv
- Cs, Os
- Cl, Ol
- ...

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Storage class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cs</td>
<td>color</td>
<td>varying</td>
<td>Surface Color described on the RIB file</td>
</tr>
<tr>
<td>Os</td>
<td>color</td>
<td>varying</td>
<td>Surface opacity described on the RIB file</td>
</tr>
<tr>
<td>P</td>
<td>point</td>
<td>varying</td>
<td>Position of shaded surface</td>
</tr>
<tr>
<td>dPdu</td>
<td>vector</td>
<td>varying</td>
<td>Derivative (tangent) of the surface position along u</td>
</tr>
<tr>
<td>dPdv</td>
<td>vector</td>
<td>varying</td>
<td>Derivative (tangent) of the surface position along v</td>
</tr>
<tr>
<td>N</td>
<td>normal</td>
<td>varying</td>
<td>Surface shading normal</td>
</tr>
<tr>
<td>Ng</td>
<td>normal</td>
<td>varying</td>
<td>Surface geometric normal</td>
</tr>
<tr>
<td>u,v</td>
<td>float</td>
<td>varying</td>
<td>Surface parameters</td>
</tr>
<tr>
<td>du, dv</td>
<td>float</td>
<td>varying</td>
<td>Change in surface parameters</td>
</tr>
<tr>
<td>s,t</td>
<td>float</td>
<td>varying</td>
<td>Surface texture coordinates</td>
</tr>
<tr>
<td>L</td>
<td>vector</td>
<td>varying</td>
<td>Incoming light ray direction*</td>
</tr>
<tr>
<td>Ci</td>
<td>color</td>
<td>varying</td>
<td>Incoming light ray color *</td>
</tr>
<tr>
<td>Oi</td>
<td>color</td>
<td>varying</td>
<td>Incoming light ray opacity*</td>
</tr>
<tr>
<td>E</td>
<td>point</td>
<td>uniform</td>
<td>Position of the eye or camera</td>
</tr>
<tr>
<td>I</td>
<td>vector</td>
<td>varying</td>
<td>Incident ray direction. Direction vector going from</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the camera to the current shading point</td>
</tr>
<tr>
<td>ncomps</td>
<td>float</td>
<td>uniform</td>
<td>Number of color components</td>
</tr>
<tr>
<td>time</td>
<td>float</td>
<td>uniform</td>
<td>Current shutter time</td>
</tr>
<tr>
<td>dtime</td>
<td>float</td>
<td>uniform</td>
<td>Amount of time covered by this shading sample</td>
</tr>
<tr>
<td>dPdtime</td>
<td>vector</td>
<td>varying</td>
<td>How the surface position P is changing per unit time, as described by motion blur in the scene.</td>
</tr>
<tr>
<td>Ci</td>
<td>color</td>
<td>varying</td>
<td>Shader output color</td>
</tr>
<tr>
<td>Oi</td>
<td>color</td>
<td>varying</td>
<td>Shader output opacity</td>
</tr>
</tbody>
</table>

* Available only in illuminance statements
Three sources of shader data

- Internal variables
- Global data from RenderMan
- User-defined parameter values via RIB file
surface showN()
{
    normal NN = normalize(N);
    vector posNorm = 0.5*(vector(1,1,1)+NN);

    Oi = Os;
    Ci = Oi*color(comp(posNorm,0), comp(posNorm,1),
                  comp(posNorm,2));
} // showN()
surface tex(string tmap="generic.tex");
{
    float alpha;

    /* get base color from map */
    if(tmap!=""")
    {
        color Ct = color texture(tmap,s,t);
        alpha = texture(tmap[3],s,t);
        Oi = alpha;
        Ci = Oi*Ct;
    }
}
} //tex()
displacement sinewaves(float freq=1.0, ampl=1.0, sphase=0, tphase=0, paramdir=0) {
    // displace along normal, using sin(s),sin(t) or both
    if(0==paramdir) {
        P += ampl*sin(sphase+s*freq*2*PI)*normalize(N);
    }
    else if (1==paramdir) {
        P += ampl*sin(tphase+t*freq*2*PI)*normalize(N);
    }
    else {
        P += ampl*sin(sphase+s*freq*2*PI)*
            sin(tphase+t*freq*2*PI)*normalize(N);
    }
    N = calculatenormal(P);
} // sinewaves
light
Kessonlt(
    float intensity=1;
    color lightcolor=1;
    float freq=1.0, coneangle=PI/2;
)
{
    point Pt = freq*transform("shader",Ps);
    vector ldir = 2*noise(freq*Pt) - 1;
    solar(ldir,coneangle)
    {
        Cl = intensity * lightcolor;
    }
}// Kessonlt()
volume underwater(
    float mindist=0, maxdist=1;
    color fg=1, bg=1;
    float inten=1, gam=1, mixf=0.5;
)
{
    color c; float d;
    d = length(I);
    if(d<=mindist)c = fg;
    else if(d>=maxdist)c = bg;
    else
    {
        d = (d-mindist)/(maxdist-mindist);
        d = pow(d,gam);
        c = mix(fg,bg,d);
    }
    Ci = inten*mix(Ci,c,mixf);
    Oi = mix( Oi, color (1,1,1), d );
}// underwater()
imager Imager_ramp
{
  color ctop = color(1,1,1);
  color cbot = color(0,0,0);
  float gam=1.0;
}
{
  float curr_y;
  float rez[3];
  color rampcol;
  float mixf;

  option("Format",rez);
  curr_y = ycomp(P)/ rez[1]; // 0 to 1, top to bottom
  curr_y = pow(curr_y,gam);

  rampcol = mix(ctop,cbot,curr_y);
  Ci += (1-Oi)*rampcol;
  Oi = 1.0;
}// Imager_ramp()
Resources

Books

Also:
• Texturing and Modeling
• Essential RenderMan
• a new RSL book
Resources

Notes, tutorials

- SIGGRAPH course notes
  '92, '95, '98, '99, '00, '01, '02, '03, '06
- Prof. Malcolm Kesson – 'fundza'
- Zhang Jian - “ZJ”
- Steve May - RManNotes
- Katsuaki Hiramitsu - 'Katsu's RooM'
Resources

Portals, forums

- RenderMan Repository – RMR – Tal Lancaster
- RenderMan Academy – Rudy Cortes
- RenderMania – Simon Bunker
- highend3d.com
- deathfall.com
- comp.graphics.rendering.renderman
Resources

Implementors' sites

- Pixar - www.pixar.com
- RenderDotC - www.dotcsw.com
- Aqsis - www.aqsis.org
- AIR - www sitiexgraphics.com
- Pixie - sourceforge.net/projects/pixie
- 3Delight - www.3delight.com
- ART - www.art-render.com/
Summary

- origins
- RI Spec
- workflow
- RIB files
- shaders (RSL)
- resources

HAVE FUN!!