Programmable Shaders

December 25, 2006

RenderMan & Its Shading Language

Key Idea of a Shading Language

- Image synthesis can be divided into two basic concerns
  - Shape: Geometric Objects, Coordinates, Transformations, Hidden-Surface Methods…
  - Shading: Light, Surface, Material, Texture, …
- Control shading not only by adjusting parameters and options, but by *telling the shader what you want it to do directly* in the form of a procedure

Pixar's RenderMan

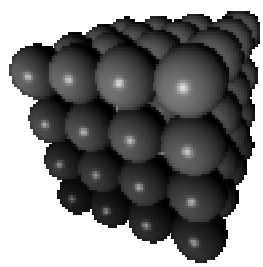
- Separation of Modeling and Rendering
  - RenderMan serves as the interface.
- Scene = Shape + Shading
- The power of RenderMan is in the shading part.

Example #1: Shape Description

```
#include <ri.h>
RtPoint Square[4]={{.5,.5,.5},{.5,-.5,.5},
          {-.5,-.5,.5},{-.5,.5,.5}};
Main(){
   RiBegin(RI_NULL);
   RiWorldBegin();
      RiSurface("constant", RI_NULL);
      RiPolygon(4, RI_P, (RtPointer)Square, RI_NULL);
   RiWorldEnd();
   RiEnd();
}
```

Example #2: Shading

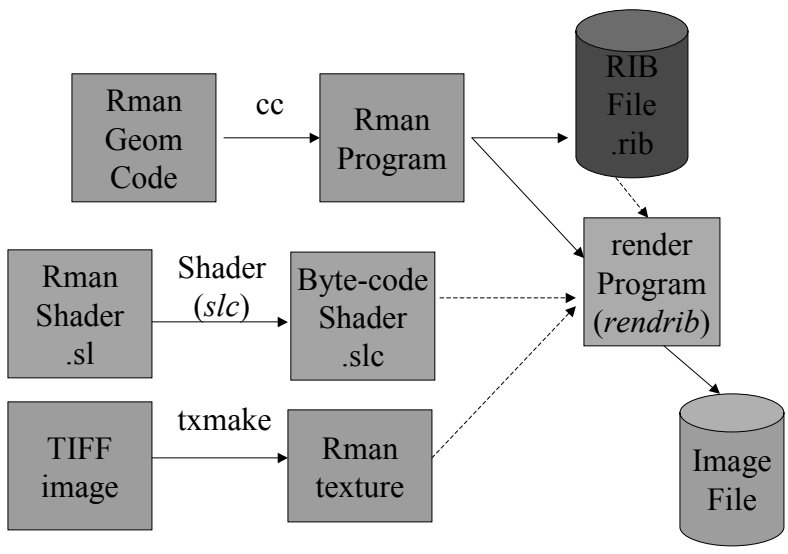
- Colorspheres.c in the BMRT/examples folder.



Building and Running

- Must have the following:
  - Header file: ri.h
  - Link library
  - A renderer
- The program generates a “RenderMan Interface” file, but doesn’t render it
  - So that you may pick a renderer that matches the graphics power of your machine.

Pixar’s RenderMan



RenderMan Interface Spec

•Where do you find the meaning of the arguments to those Ri…() functions?

- Check the spec! (available directly from Pixar).
- Appendix G contains a quick reference.

http://www.pixar.com/renderman/developers_corner/rispec/rispec_pdf/RISpec3_2.pdf

Shading Language

- Many types of shaders are possible:
  - Light source shaders
  - Surface shaders
  - Atmosphere shaders
  - Volume shaders…etc.
- We will discuss only the surface shaders.

Shader Writing

- Global variables: (from Table 14.2 of *The RenderMan Companion* book)
  - Camera position, surface color/opacity, surface position/normal, texture coordinates…etc.
  - Must output: color of light from surface, opacity of surface.
- Many built-in operators (Table 14.4) and functions (Ch.15, Tables 15.1-15.2).

Example: Plastic Surface

```
surface
plastic (float Ka = 1, Kd = 0.5, Ks = 0.5,
                roughness = 0.1;
         color specularcolor = 1)
{
    normal Nf = faceforward (normalize(N),I);
    Ci = Cs * (Ka*ambient() + Kd*diffuse(Nf)) +
         specularcolor *
         Ks*specular(Nf,-normalize(I),roughness);
    Oi = Os;  Ci *= Oi;
}
```

RenderMan's Shader

- Phong shader

```
surface phong( float Ka = 1, Kd =1, Ks = 0.5;
        float roughness = 0.1;
       color specularcolor = 1; )
{
   normal Nf = faceforward( normalize(N), I );
   vector V = -normalize(I);
   color C = 0;
   illuminance( P ) {
      vector R = 2*normalize(N)*
                (normalize(N) . normalize( L )) -
                 normalize( L );
      C += Ka*Cs +
           Kd*Cs*( normalize(N) . normalize(L) ) +
           Ks*specularcolor* pow(( R . V ), 10);
   }
   Ci = C*Cs;
}
```

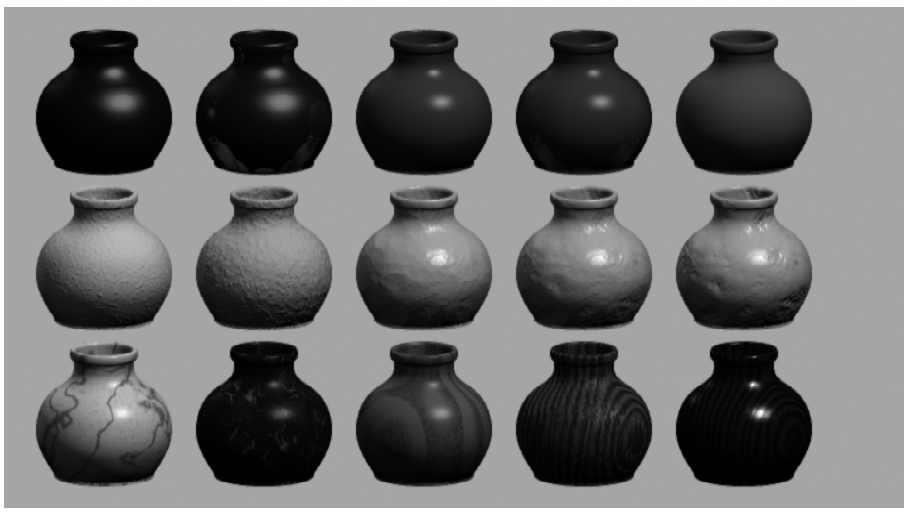
RenderMan's Shader -2

- Attaching to the RIB file

```
***********
AttributeBegin
Translate 0 -1.5 0
Rotate 20 0 0 1
Color [ 0.8 0.0 0.0 ]
Surface "phong" "Ka" [.1] "Kd" [.8] "Ks" [1]
     "roughness" [0.1] "specularcolor" [1 1 1]
Basis "bezier" 3 "bezier" 3
PatchMesh "bicubic" 13 "nonperiodic" 10
"nonperiodic" "P" [1.5 0 0 1.5 0.828427 0 0.828427
1.5 0 0 1.5 0 -0.828427 1.5 0 -1.5 0.828427 0 -1.5
0 0 -1.5 -0.828427 0 -0.828427 -1.5 0 0 -1.5 0
0.828427 -1.5 0 1.5 -0.828427 0 1.5 0 0 1.5 0 0.075
1.5 0.828427 0.075 0.828427 1.5 0.075 0 1.5 0.075 -
0.828427 1.5 0.075 -1.5 0.828427 0.075 -1.5 0 0.075
-1.5 -0.828427 0.075 -0.828427 -1.5 0.075 0 -1.5
0.075 0.828427 -1.5 0.075 1.5 -0.828427 0.075 1.5 0
0.075 2 0 0.3 2 1.10457 0.3 1.10457 2 0.3 0 2 0.3 -
1.10457
**************
```

Gallery of Shaders

- Procedural textures: e.g., wood
- Bump mapping and displacement mapping.
- For more, see Ch. 16 of *The RenderMan Companion*
- See also the `shaders` and `examples` folders of BMRT.



A Few Stories…

- The shader concept first appears in the REYES paper.
- The programmable shader first appears in graphics hardware in UNC's PixelFlow [Olano97]. Many folks in the PixelFlow team are now at NVIDIA.
- The co-author of PBRT and the creator of BMRT (Larry Gritz) were cofounders of Exluna

REYES

- From Cook et al. "The Reyes Image Rendering Architecture" SIGGRAPH 87.
- Subdivide a surface into micropolygons.

BMRT

- A public-domain implementation of Pixar Photorealistic RenderMan (PRMan).
- Three main components:
  - Rendrib: the renderer
  - Rgl: quick rendering for preview
  - Slc: shading language compiler

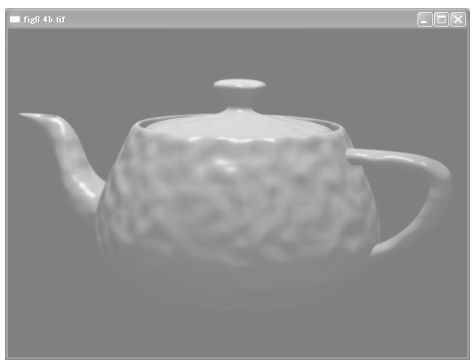
Shading Languages for Graphics Hardware

A Brief Introduction to Cg

- Cg is "C for Graphics," a high-level, cross-platform language for graphics programming
- Cg replaces tedious assembly coding with a C-like language and a compiler that generates assembly for you
- Cg is a cross-API and cross-platform language:
  - It works with both OpenGL and DirectX
  - It runs on Windows and Linux (more in the works)
  - It supports hardware from NVIDIA, ATI, Matrox, and any other programmable hardware that supports OpenGL or DirectX
- The Cg Runtime simplifies parameter passing from your application to the vertex and fragment programs

Bump Map Example

- Bump mapping simulates detail with a surface normal that varies across a surface

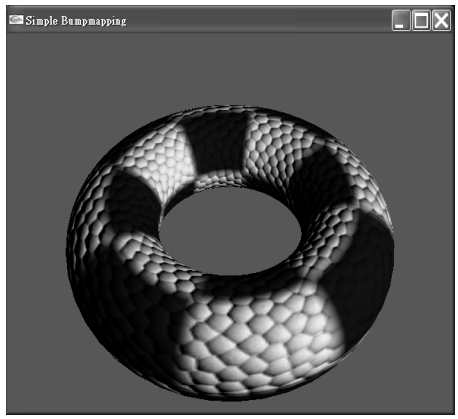


RenderMan Example

```
displacement
lumpy ( float Km = 1, frequency = 1, maxoctaves = 6;
        string shadingspace = "shader";
        float truedisp = 1;)
{
   point Pshad = transform (shadingspace, frequency*P);
   float dPshad = filterwidthp(Pshad);
   float magnitude = fBm (Pshad, dPshad, maxoctaves, 2, 0.5);
   N = Displace (normalize(N), shadingspace, Km*magnitude, truedisp);
}
```


Cg Example

```
f2fb DiffuseBumpPS(v2f IN, uniform sampler2D DiffuseMap,
                   uniform sampler2D NormalMap, uniform float4 bumpHeight)
{
    f2fb OUT;
    float4 color = tex2D(DiffuseMap); //fetch base color
    //fetch bump normal
    float4 bumpNormal = expand(tex2D(NormalMap)) * bumpHeight;
    //expand iterated light vector to [-1,1]
    float4 lightVector = expand(passthrough(IN.LightVector));
    //compute final color (diffuse + ambient)
    float4 bump = uclamp(dot3_rgba(bumpNormal.xyz, lightVector.xyz));
    OUT.col = color * bump;
    return OUT;
}
```


Simple Bumpmapping