Lecture 3:
Shadow and Environment Mapping
Recap: Last Lecture

- A simple material: Blinn-Phong reflectance model
- Basic GPU hardware pipeline
- Shading languages
- The rendering equation
Today’s Lecture
Outline

• Shading languages

• The rendering equation

• Basic shadow mapping techniques

• Environment mapping
Shading Languages

- Vertex / Fragment shading described by small program
- Written in language similar to C but with restrictions
- Long history. Cook’s paper on Shade Trees, Renderman for offline rendering

- Stanford Real-Time Shading Language, work at SGI
- Cg from NVIDIA
- HLSL in DirectX (vertex + pixel)
- **GLSL** directly compatible with OpenGL 2.0 (So, you can just read the OpenGL Red Book to get started)
Shader Setup

• Initializing (shader itself discussed later)
  - Create shader (Vertex and Fragment)
  - Compile shader
  - Attach shader to program
  - Link program
  - Use program

• Shader source is just sequence of strings

• Similar steps to compile a normal program
Shader Initialization Code

```c
GLuint initshaders (GLenum type, const char *filename) {
    // Using GLSL shaders, OpenGL book, page 679
    GLuint shader = glCreateShader(type) ;
    GLint compiled ;
    string str = textFileRead (filename) ;
    GLchar * cstr = new GLchar[str.size()+1] ;
    const GLchar * cstr2 = cstr ; // Weirdness to get a const char
    strcpy(cstr,str.c_str()) ;
    glShaderSource (shader, 1, &cstr2, NULL) ;
    glCompileShader (shader) ;
    glGetShaderiv (shader, GL_COMPILE_STATUS, &compiled) ;
    if (!compiled) {
        shadererrors (shader) ;
        throw 3 ;
    }
    return shader ;
}
```
Linking Shader Program

GLuint initprogram (GLuint vertexshader, GLuint fragmentshader)
{
    GLuint program = glCreateProgram() ;
    GLint linked ;
    glAttachShader(program, vertexshader) ;
    glAttachShader(program, fragmentshader) ;
    glLinkProgram(program) ;
    glGetProgramiv(program, GL_LINK_STATUS, &linked) ;
    if (linked) glUseProgram(program) ;
    else {
        programerrors(program) ;
        throw 4 ;
    }
    return program ;
}
Phong Shader: Vertex

This Shader Does
- Gives eye space location for \( v \)
- Transform Surface Normal
- Transform Vertex Location

```glsl
varying vec3 N;
varying vec3 v;

void main(void)
{
    v = vec3(gl_ModelViewMatrix * gl_Vertex);
    N = normalize(gl_NormalMatrix * gl_Normal);

    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Cliff Lindsay web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/gpu.pdf
varying vec3 N;
varying vec3 v;

void main (void)
{
    // we are in Eye Coordinates, so EyePos is (0,0,0)
    vec3 L = normalize(gl_LightSource[0].position.xyz - v);
    vec3 E = normalize(-v);
    vec3 R = normalize(-reflect(L,N));

    //calculate Ambient Term:
    vec4 lamb = gl_FrontLightProduct[0].ambient;

    //calculate Diffuse Term:
    vec4 ldiff = gl_FrontLightProduct[0].diffuse * max(dot(N,L), 0.0);

    // calculate Specular Term:
    vec4 lspec = gl_FrontLightProduct[0].specular
        * pow(max(dot(R,E),0.0), gl_FrontMaterial.shininess);

    // write Total Color:
    gl_FragColor = gl_FrontLightModelProduct.sceneColor + lamb + ldiff + lspec;
}
Questions?
Outline

• Shading languages
• The rendering equation
• Basic shadow mapping techniques
• Environment mapping
Reflection Equation

\[ L_r(x, \omega_r) = L_e(x, \omega_r) + L_i(x, \omega_i) \cdot f(x, \omega_i, \omega_r)(\omega_i, n) \]

- Reflected Light (Output Image)
- Emission
- Incident Light (from light source)
- BRDF
- Cosine of Incident angle
Reflection Equation

\[ L_r(x, \omega_r) = L_e(x, \omega_r) + \sum L_i(x, \omega_i) f(x, \omega_i, \omega_r) (\omega_i, n) \]

Reflected Light (Output Image)  Emission  Incident Light (from light source)  BRDF  Cosine of Incident angle

Sum over all light sources
Reflection Equation

\[ L_r(x, \omega_r) = L_e(x, \omega_r) + \int_{\Omega} L_i(x, \omega_i) f(x, \omega_i, \omega_r) \cos \theta_i \, d\omega_i \]

- **Reflected Light (Output Image)**
- **Emission**
- **Incident Light (from light source)**
- **BRDF**
- **Cosine of Incident angle**

Replace sum with integral
Rendering Equation

Surfaces (interreflection)

\[ \mathbf{L}_r(\mathbf{x}, \omega_r) = \mathbf{L}_e(\mathbf{x}, \omega_r) + \int_{\Omega} \mathbf{L}_r(\mathbf{x}', -\omega_i) f(\mathbf{x}, \omega_i, \omega_r) \cos \theta_i d\omega_i \]

Reflected Light (Output Image)
Emission
Reflected Light
BRDF
Cosine of Incident angle

UNKNOWN
KNOWN
UNKNOWN
KNOWN
KNOWN
Rendering Equation (Kajiya 86)

Figure 6. A sample image. All objects are neutral grey. Color on the objects is due to caustics from the green glass balls and color bleeding from the base polygon.
Rendering Equation as Integral Equation

\[ L_r(x, \omega_r) = L_e(x, \omega_r) + \int_{\Omega} L_r(x', -\omega_i) f(x, \omega_i, \omega_r) \cos \theta_i d\omega_i \]

Reflected Light (Output Image)  Emission  Reflected Light  BRDF  Cosine of Incident angle

UNKNOWN  KNOWN  UNKNOWN  KNOWN  KNOWN

Is a Fredholm Integral Equation of second kind [extensively studied numerically] with canonical form

\[ l(u) = e(u) + \int l(v) K(u, v) dv \]

Kernel of equation
Linear Operator Equation

\[ l(u) = e(u) + \int l(v) K(u, v) dv \]

Kernel of equation
Light Transport Operator

\[ L = E + KL \]

Can be discretized to a simple matrix equation
[or system of simultaneous linear equations]
(L, E are vectors, K is the light transport matrix)
Ray Tracing and extensions

- General class numerical Monte Carlo methods
- Approximate set of all paths of light in scene

\[ L = E + KL \]

\[ LL - KL = E \]

\[ (I - K)L = E \]

\[ L = (I - K)^{-1}E \]

**Binomial Theorem**

\[ L = (I + K + K^2 + K^3 + \ldots)E \]

\[ L = E + KE + K^2E + K^3E + \ldots \]
Ray Tracing

\[ L = E + KE + K^2E + K^3E + \ldots \]

- Emission directly from light sources
- Direct Illumination on surfaces
- Indirect Illumination (One bounce indirect) [Mirrors, Refraction]
- (Two bounce indirect illum.)
Ray Tracing

\[ L = E + KE + K^2E + K^3E + \ldots \]

- Emission directly from light sources
- Direct Illumination on surfaces
- Indirect Illumination (One bounce indirect) [Mirrors, Refraction]
- (Two bounce indirect illum.)

OpenGL Shading
Direct illumination
One-bounce global illumination
Two-bounce global illumination
Four-bounce global illumination
Eight-bounce global illumination
Sixteen-bounce global illumination
Questions?
Outline

- Shading languages
- The rendering equation
- Basic shadow mapping techniques
- Environment mapping
Shadow and Environment Maps

- Basic methods to **add realism** to interactive rendering
- **Shadow maps**: image-based way, hard shadows
  - Very old technique. Originally Williams 78
  - Many recent (and older) extensions
  - Widely used even in software rendering (RenderMan)
  - Simple alternative to raytracing for shadows
- **Environment maps**: image-based complex lighting
  - Again, very old technique. Blinn and Newell 76
  - Huge amount of recent work (some covered in course)
- Together, give most of realistic effects we want
  - **But cannot be easily combined!!** Some of the course is about ways to get around this limitation
  - See Annen 08 [real-time all-frequency shadows dynamic scenes] for one approach: convolution soft shadows
Common Real-time Shadow Techniques

- Projected planar shadows
- Shadow volumes
- Light maps

This slide, others courtesy Mark Kilgard
Problems

Mostly tricks with lots of limitations

- Projected planar shadows
  works well only on flat surfaces

- Stenciled shadow volumes
  determining the shadow volume is hard work

- Light maps
  totally unsuited for dynamic shadows

- In general, hard to get everything shadowing everything
Shadow Mapping

- Lance Williams: Brute Force in image space (shadow maps in 1978, but other similar ideas like Z buffer, bump mapping using textures and so on)

- Completely image-space algorithm
  - no knowledge of scene’s geometry is required
  - must deal with aliasing artifacts

- Well known software rendering technique
  - Basic shadowing technique for Toy Story, etc.
Phase 1: Render from Light

- Depth image from light source
Phase 1: Render from Light

- Depth image from light source
Phase 2: Render from Eye

- Standard image (with depth) from eye
Phase 2+: Project to light for shadows

- Project visible points in eye view back to light source

(Reprojected) depths match for light and eye. VISIBLE
Phase 2+: Project to light for shadows

- Project visible points in eye view back to light source

(Reprojected) depths from light, eye not the same. BLOCKED!!
Visualizing Shadow Mapping

- A fairly complex scene with shadows
Visualizing Shadow Mapping

- Compare with and without shadows

with shadows  without shadows
Visualizing Shadow Mapping

- The scene from the light's point-of-view

*FYI: from the eye's point-of-view again*
Visualizing Shadow Mapping

- The depth buffer from the light’s point-of-view

FYI: from the light’s point-of-view again
Visualizing Shadow Mapping

- Projecting the depth map onto the eye’s view

FYI: depth map for light’s point-of-view again
Visualizing Shadow Mapping

- Comparing light distance to light depth map

Green is where the light planar distance and the light depth map are approximately equal.

Non-green is where shadows should be.
Visualizing Shadow Mapping

- Scene with shadows

Notice how specular highlights never appear in shadows

Notice how curved surfaces cast shadows on each other
Problems with shadow maps

- Hard shadows (point lights only)
- Quality depends on shadow map resolution (general problem with image-based techniques)
- Involves equality comparison of floating point depth values means issues of scale, bias, tolerance
- Some of these addressed in papers presented
Questions?