CSE167
Introduction to Computer Graphics

Matthias Zwicker
University of California, San Diego
Spring 2008

Today
Tricks for more realistic shading
• Shading using environment maps
• Reflection and refraction
• Soft shadows

Shading using environment map
• Assumption: distant lighting
  - Incident light is a function of direction, but not position
• Realistic shading requires
  - Take into account light from all directions
  - Include occlusion

Mathematical model
• Assume Lambertian material
• Ignore occlusion for now
• Illumination from point light sources
  \[ c = \sum_i c_i k_d (L_i \cdot n) \]
• Illumination from environment map
  \[ c = k_d \int_{\Omega} c(\omega)(\omega \cdot n) d\omega \]
  - Directions \( \omega \)
  - Hemisphere of directions \( \Omega \)
  - Environment map \( c(\omega) \)

Irradiance environment maps
• Precompute irradiance as a function of normal
  \[ B(n) = \int_{\Omega} c(\omega)(\omega \cdot n) d\omega \]
• Store as irradiance environment map
• Shading computation at render time
  \[ c = k_d B(n) \]

Irradiance environment maps
• Directional light
• Environment illumination
**Implementation**

- Precompute irradiance map from environment
  - HDRShop tool, “diffuse convolution”
    http://projects.ict.usc.edu/graphics/HDRShop/
- At render time, look up irradiance map using surface normal
- Can also approximate glossy reflection
  - Blur environment map less heavily
  - Look up blurred environment map using reflection vector

**Including occlusion**

Visibility function $V_x$

- Binary function of direction
- Indicates if environment is occluded
- Depends on position $x$

![Environment map](image)

- $V_x = 0$
- $V_x = 0$

**Mathematical model**

- Include visibility function
  \[ c = k_i \int V_x(\omega) c(\omega \cdot n) d\omega \]
- **Ambient occlusion**
  - “Fraction” of environment that is not occluded from a point $x$
  - Scalar value
    \[ a_x = \int V_x(\omega) (\omega \cdot n) d\omega \]
- Approximation
  \[ c = k_a a_x k_i \]

**Ambient occlusion**

- ![Ambient occlusion](image)
- ![Diffuse shading](image)
- ![Ambient occlusion combined with diffuse shading](image)

**Implementation**

- Precomputation
  - Compute ambient occlusion on a per-vertex basis
  - Ray tracing
  - Free tool that saves meshes with per-vertex ambient occlusion
    http://www.xnormal.net/
- Caution
  - Ambient occlusion does not work for animated objects

**Today**

**Tricks for more realistic shading**

- Shading using environment maps
- Reflection and refraction
- Soft shadows
Reflection & refraction

Refraction

- Light rays that travel from one medium to another are bent
- To the viewer, object at location \( x \) appears to be at location \( y \)

Water

Air

Index of refraction

- Speed of light depends on medium
  - Speed of light in vacuum \( c \)
  - Speed of light in medium \( v \)
- Index of refraction \( n = c/v \)
  - Air 1.00029
  - Water 1.33
  - Acrylic glass 1.49
- “Change in phase velocity leads to bending of light rays”

Snell’s law

- Ratio of sines angles of incidence and refraction is equal to opposite ratio of indices of refraction
  \[
  \frac{\sin \theta_1}{\sin \theta_2} = \frac{n_2}{n_1}
  \]
- Vector form
  \[
  \mathbf{r} = \frac{n_1}{n_2} \mathbf{v} + \left( \frac{n_1}{n_2} \cos \theta_2 + \cos \theta_2 \right) \mathbf{n}
  \]
  - Viewing, refracted direction \( \mathbf{v}, \mathbf{r} \)
  - Normal vector \( \mathbf{n} \)

Total internal reflection

- Angle of refracted ray
  \[
  \theta_2 = \arcsin \left( \frac{n_1 \sin \theta_1}{n_2} \right)
  \]
- Critical angle
  \[
  \theta_c = \frac{n_2}{n_1}
  \]
  - If \( \theta_1 = \theta_c \) we get \( \theta_2 = \pi/2 \), refracted ray is parallel to interface
  - If \( \theta_1 > \theta_c \) we have total internal reflection

Fresnel equations

- When light travels from one medium to another, both reflection and refraction may occur
- Fresnel equations describe fraction of intensity of light that is reflected and refracted
  - Depend on polarization of light
**Fresnel equations**

- Fresnel equations are relatively complex to evaluate.
- In graphics, often use Schlick’s approximation.
  - Ratio $F$ between reflected and refracted light.
  
  \[
  F = f + (1 - f)(1 - v \cdot n)^5
  \]
  \[
  f = \frac{(1.0 - n_1)}{(1.0 + n_2)}
  \]

**Implementation**

- Accurate implementation requires ray tracing.
- For interactive graphics, approximation using environment maps.
  - Use reflected, refracted rays to look up environment map.
  - Does not take into account geometry after first bounce.
  - Assumes illumination is infinitely far away.

**Vertex shader**

```shaderv
const float Eta = 0.67; // Ratio of indices of refraction (air -> glass)
const float FresnelPower = 10.0; // Controls degree of reflectivity at grazing angles
const float F  = ((1.0 - Eta) * (1.0 - Eta)) / ((1.0 + Eta) * (1.0 + Eta));

varying vec3  Reflect;
varying vec3  Refract;
varying float Ratio;

void main(void)
{
  vec4 ecPosition = gl_ModelViewMatrix * gl_Vertex;
  vec3 ecPosition3 = ecPosition.xyz / ecPosition.w;
  vec3 i = normalize(ecPosition3);
  vec3 n = normalize(gl_NormalMatrix * gl_Normal);
  Ratio = F + (1.0 - F) * pow((1.0 - dot(-i, n)), FresnelPower);
  Refract = refract(i, n, Eta);
  Refract = vec3(gl_TextureMatrix[0] * vec4(Refract, 1.0));
  Reflect = reflect(i, n);
  Reflect = vec3(gl_TextureMatrix[0] * vec4(Reflect, 1.0));
  gl_Position = ftransform();
}
```

**Fragment shader**

```shaderv
varying vec3  Reflect;
varying vec3  Refract;
varying float Ratio;
uniform samplerCube cubemap;

void main(void)
{
  vec3 refractColor = vec3 (textureCube(cubemap, Refract));
  vec3 reflectColor = vec3 (textureCube(cubemap, Reflect));
  vec3 color   = mix(refractColor, reflectColor, Ratio);
  gl_FragColor = vec4(color, 1.0);
}
```

**Chromatic dispersion**

- Phase velocity in many media depends on wavelength/frequency.
  - Dispersive media.
  - Different colors refract at different angles.

- In the context of camera lenses, chromatic aberration.
  - Try to avoid using achromatic lenses.
Implementation

• Approximate dispersion by using three different ratios of indices of refraction for R,G,B channels
  - Glass: 0.65, 0.67, 0.69
• Perform separate look-ups for R,G,B channels in environment map

Today

Tricks for more realistic shading

• Shading using environment maps
• Reflection and refraction
• Soft shadows

Shadow volumes

Shading with shadow volumes

• Many variations
• Stencil shadow volumes
  - Classic algorithm
  - Hard shadows
• Here, two-pass algorithm for approximate soft shadows
  - Very simple and inaccurate, but often plausible enough
• Many more complicated and more accurate variations exist

Shadow volume construction

• Need to generate shadow polygons to bound shadow volume
• Extrude silhouette edges from light source

Shadow volume construction

• Needs to be done on the CPU
• Silhouette edge detection
  - An edge is a silhouette if one adjacent triangle is front facing, the other back facing with respect to the light
• Extrude polygons from silhouette edges
**Two-pass algorithm**

First pass
- Render scene without shading, store depth buffer

Second pass
- Use depth image from first pass for z-buffering
- Render polygons of shadow volume only
- For each rasterized pixel, compute a fractional shadow value in a fragment shader

**Resources**
- OpenGL Shading Language book
- Webpage for the book [http://3dshaders.com](http://3dshaders.com)
- Has example code and data

**Two-pass algorithm**

Approximation
- Shadow value depends on distance to center of occluder, distance to center of shadow volume

Second pass
- Construct a local coordinate system for each shadow volume
  - Origin is center of occluder
  - z-axis is center of shadow volume
- Transform location of each pixel to shadow volume coordinate system
- Use z-coordinate and distance to z-axis to look up shadow texture