Global Illumination

Jason Lawrence

CS 4810: Graphics

Acknowledgment: slides by Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin
Overview

• Direct Illumination
  • Emission at light sources
  • Direct light at surface points

• Global illumination
  • Shadows
  • Transmissions
  • Inter-object reflections
Shadows

• Shadow term tells if light sources are blocked
  ◦ Cast ray towards each light source $L_i$. If the ray is blocked, do not consider the contribution of the light.
Shadows

- Shadow term tells if light sources are blocked
  - Cast ray towards each light source $L_i$
  - $S_i = 0$ if ray is blocked, $S_i = 1$ otherwise

\[
I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L
\]
Shadows

- Shadow term tells if light sources are blocked
  - Cast ray towards each light source $L_i$
  - $S_i = 0$ if ray is blocked, $S_i = 1$ otherwise

$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L$$
Shadows

• Shadow term tells if light sources are blocked
  Cast ray towards each light source $L_i$
  $S_i = 0$ if ray is blocked, $S_i = 1$ otherwise

$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L$
Ray Casting

- Trace primary rays from camera
  - Direct illumination from unblocked lights only
Recursive Ray Tracing

• Also trace secondary rays from hit surfaces
  o Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays
Mirror Reflections

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[
I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R
\]
Mirror Reflections

• Also trace secondary rays from hit surfaces
  o Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[ I = I_E + K_A I_A + \sum_{L} (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R \]
Mirror Reflections

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[ I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R \]
Mirror Reflections

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[
I = I_E + K_A I_A + \sum L \left( K_D (N \cdot L) + K_S (V \cdot R)^n \right) I_L S_L + K_S I_R
\]
Mirror Reflections

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[
I = I_E + K_A I_A + \sum_L (K_D (\mathbf{N} \cdot \mathbf{L}) + K_S (\mathbf{V} \cdot \mathbf{R})^n) I_L S_L + K_S I_R
\]
Mirror Reflections

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays
Transparent Refraction

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[
I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T
\]
Transparent Refraction

• Also trace secondary rays from hit surfaces
  ○ Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[ I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T \]
Transparent Refraction

• Also trace secondary rays from hit surfaces
  o Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[
I = I_E + K_A I_A + \sum_L \left( K_D (N \cdot L) + K_S (V \cdot R)^n \right) I_L S_L + K_S I_R + K_T I_T
\]
Transparent Refraction

• Also trace secondary rays from hit surfaces
  o Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[ I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T \]
Transparent Refraction

• Also trace secondary rays from hit surfaces
  o Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

\[
I = I_E + K_A I_A + \sum_L \left( K_D (N \cdot L) + K_S (V \cdot R)^n \right) I_L S_L + K_S I_R + K_T I_T
\]
Transparent Refraction

• Also trace secondary rays from hit surfaces
  ○ Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays
Transparency and Shadow

Problem:
- If a surface is transparent, then rays to the light source may pass through the object.
Transparency and Shadow

- Problem:
  - If a surface is transparent, then rays to the light source may pass through the object.
  - Need to modify the shadow term so that instead of representing a binary (0/1) value, it gives the fraction of light passing through.

\[
I = I_E + K_A I_A + \sum L (K_D (N \cdot L) + K_S (V \cdot R)\^n) I_L (S_L) + K_S I_R + K_T I_T
\]
Transparency and Shadow

• Problem:
  o If a surface is transparent, then rays to the light source may pass through the object.
  o Need to modify the shadow term so that instead of representing a binary (0/1) value, it gives the fraction of light passing through.
  o Accumulate transparency values as the ray travels to the light source.

\[ I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T \]
Transparency and Shadow

• Accumulate transparency values as the ray travels to the light source.

\[ I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T \]
Transparency and Shadow

• Accumulate transparency values as the ray travels to the light source.

\[ I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T \]

\[ S_0 = 1: \]
• \( L_0 \) contributes fully
\[ S_I = ? \]
Transparency and Shadow

• Accumulate transparency values as the ray travels to the light source.

\[ I = I_E + K_A I_A + \sum L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T \]
Transparency and Shadow

• Accumulate transparency values as the ray travels to the light source.

\[
I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T
\]

\[S_0=1: \quad L_0 \text{ contributes fully}\]
\[S_i=1*K_T*K_T^*?\]
Transparency and Shadow

• Accumulate transparency values as the ray travels to the light source.

\[ I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T \]
Transparency and Shadow

- Accumulate transparency values as the ray travels to the light source.
Transparent Refraction

• When a ray of light passes through a transparent object it can bend.
Transparent Refraction

• When a ray of light passes through a transparent object, the ray of light can bend, ($\theta_i \neq \theta_r$).

![Diagram showing refraction of light](image-url)
**Snell’s Law**

- The way that light bends is determined by the indices of refraction of the internal and external materials $\eta_i$ and $\eta_r$:

\[
\eta_r \sin \theta_r = \eta_i \sin \theta_i
\]

- The index of refraction of air is $\eta=1$. 
Snell’s Law

- The way that light bends is determined by the indices of refraction of the internal and external materials $\eta_i$ and $\eta_r$:

$$\eta_r \sin \theta_r = \eta_i \sin \theta_i$$

$$T = \left( \frac{\eta_i}{\eta_r} \cos \theta_i - \cos \theta_r \right) \frac{1}{N} - \frac{\eta_i}{\eta_r} L$$
Snell’s Law

- The way that light bends is determined by the indices of refraction of the internal and external materials $\eta_i$ and $\eta_r$:

\[ \eta_r \sin \theta_r = \eta_i \sin \theta_i \]
Snell’s Law

• The way that light bends is determined by the indices of refraction of the internal and external materials $\eta_i$ and $\eta_r$:

$$\eta_r \sin \theta_r = \eta_i \sin \theta_i$$
Snell’s Law

- The way that light bends is determined by the indices of refraction of the internal and external materials $\eta_i$ and $\eta_r$:

$$\eta_r \sin \theta_r = \eta_i \sin \theta_i$$
Snell’s Law

• The way that light bends is determined by the indices of refraction of the internal and external materials $\eta_i$ and $\eta_r$:

$$\eta_r \sin \theta_r = \eta_i \sin \theta_i$$
Snell’s Law

- The way that light bends is determined by the indices of refraction of the internal and external materials $\eta_i$ and $\eta_r$:

$$\eta_r \sin \theta_r = \eta_i \sin \theta_i$$
Snell’s Law

• The way that light bends is determined by the indices of refraction of the internal and external materials $\eta_i$ and $\eta_r$: 
Snell’s Law and Shadows

• Problem:
  - If a surface is transparent, then rays to the light source may not travel in a straight line
Snell’s Law and Shadows

• Problem:
  o If a surface is transparent, then rays to the light source may not travel in a straight line
  o This is difficult to address with ray-tracing
General Issue

• How do we determine when to stop recursing?
General Issue

• How do we determine when to stop recursing?
  o Depth of iteration
    » Bounds the number of times a ray will bounce around the scene
  o Cut-off value
    » Ignores contribution from bounces that contribute very little
Putting it all Together

Image RayCast(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(scene, ray, hit);
        }
    }
    return image;
}
Putting it all Together

```java
Image RayCast(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(scene, ray, hit);
        }
    }
    return image;
}
```

With Illumination
Putting it all Together

```c
Pixel GetColor(scene, ray, depth, cutOff) {
    Pixel p(0,0,0)
    Ray reflect, refract
    Intersection hit=FindIntersection(ray, scene);
    if (hit) {
        p += GetSurfaceColor(hit.position);
        
        reflect.direction = Reflect(ray.direction, hit.normal)
        reflect.position = hit.position + reflect.direction*ε
        if (depth >0 && hit.kSpec>cutOff)
            p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)*hit.kSpec
        
        refract.direction = Refract(ray.direction, hit.normal, hit.ir)
        refract.position = hit.position + refract.direction*ε
        if (depth >0 && hit.kTran>cutOff)
            p += GetColor(scene, refract, depth-1, cutOff/hit.kTran)*hit.kTran
    }
    return p
}
```
Putting it all Together

Pixel GetColor(scene, ray, depth, cutOff){
    Pixel p(0,0,0)
    Ray reflect, refract
    Intersection hit=FindIntersection(ray, scene);
    if (hit){
        p += GetSurfaceColor(hit.position);
        reflect.direction = Reflect(ray.direction, hit.normal)
        reflect.position = hit.position + refract.direction*ε
        if( depth > 0 && hit.kSpec>cutOff)
            p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)*hit.kSpec
        refract.direction = Refract(ray.direction, hit.normal, hit.ir)
        refract.position = hit.position + refract.direction*ε
        if( depth > 0 && hit.kTran>cutOff)
            p += GetColor(scene, refract, depth-1, cutOff/hit.kTran)*hit.kTran
    }
    return p
}
Putting it all Together

Pixel GetColor(scene, ray, depth, cutOff) {
    Pixel p(0,0,0)
    Ray reflect, refract
    Intersection hit=FindIntersection(ray, scene);
    if ( hit ) {
        p += GetSurfaceColor(hit.position);

        reflect.direction = Reflect(ray.direction, hit.normal)
        reflect.position = hit.position + reflect.direction * ε
        if( depth >0 && hit.kSpec>cutOff)
            p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)*hit.kSpec

        refract.direction = Refract(ray.direction, hit.normal, hit.ir)
        refract.position = hit.position + refract.direction*ε
        if( depth >0 && hit.kTran>cutOff)
            p += GetColor(scene, refract, depth-1, cutOff/hit.kTran)*hit.kTran
    }
    return p
}

\[ I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) I_L S_L + K_S I_R + K_T I_T \]
Putting it all Together

Pixel GetColor(scene, ray, depth, cutOff){
    Pixel p(0,0,0)
    Ray reflect, refract
    Intersection hit=FindIntersection(ray, scene);
    if ( hit ){
        p += GetSurfaceColor(hit.position);

        reflect.direction = Reflect( ray.direction, hit.normal)
        reflect.position = hit.position + reflect.direction *ε
        if( depth >0 && hit.kSpec>cutOff)
            p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)*hit.kSpec

        refract.direction = Refract( ray.direction, hit.normal, hit.ir)
        refract.position = hit.position + refract.direction*ε
        if( depth >0 && hit.kTran>cutOff)
            p += GetColor(scene, refract, depth-1, cutOff/hit.kTran)*hit.kTran
    }
    return p
}

\[ I = I_E + K_A I_A + \sum_L \left( K_D (N \cdot L) + K_S (N \cdot R)^n \right) I_L S_L + K_S I_R + K_T I_T \]
Putting it all Together

Pixel GetColor(scene, ray, depth, cutoff){
    Pixel p(0,0,0)
    Ray reflect, refract
    Intersection hit = FindIntersection(ray, scene);
    if ( hit ){
        p += GetSurfaceColor(hit.position);

        reflect.direction = Reflect(ray.direction, hit.normal)
        reflect.position = hit.position + reflect.direction * ε
        if( depth > 0 && hit.kSpec > cutoff)
            p += GetColor(scene, reflect, depth-1, cutoff/hit.kSpec)

        refract.direction = Refract(ray.direction, hit.normal, hit.ir)
        refract.position = hit.position + refract.direction * ε
        if( depth > 0 && hit.kTran > cutoff)
            p += GetColor(scene, refract, depth-1, cutoff/hit.kTran)
    }
    return p
}
Pixel GetColor(scene, ray, \textbf{depth}, \textbf{cutOff}){
    Pixel p(0,0,0)
    Ray reflect, refract
    Intersection hit=FindIntersection(ray, scene);
    if (hit){
        p += GetSurfaceColor(hit.position);
        reflect.direction = Reflect(ray.direction, hit.normal)
        reflect.position = hit.position + reflect.direction * \varepsilon
        if( depth >0 && hit.kSpec>cutOff)
            p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)
    }
    refract.direction = Refract(ray.direction, hit.normal, hit.ir)
    refract.position = hit.position + refract.direction*\varepsilon
    if( depth >0 && hit.kTran>cutOff)
        p += GetColor(scene, refract, depth-1, cutOff/hit.kTran)
}
return p
Illumination Examples

• Ray tracing

Courtesy Henrik Wann Jensen
Illumination Examples

• Soft Shadows

Courtesy Henrik Wann Jensen
Illumination Examples

- Caustics

Courtesy Henrik Wann Jensen
Illumination Examples

- Full Global Illumination
Recursive Ray Tracing

- **GetColor** is a recursive function

```java
Image RayTrace(Camera camera, Scene scene, int width, int height, int depth, float cutOff) {
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            image[i][j] = GetColor(scene, ray, depth, cutOff);
        }
    }
    return image;
}
```
Summary

• Ray casting (direct Illumination)
  - Usually use simple analytic approximations for light source emission and surface reflectance

• Recursive ray tracing (global illumination)
  - Incorporate shadows, mirror reflections, and pure refractions

All of this is an approximation so that it is practical to compute