Local Illumination

Greg Humphreys
CS445: Intro Graphics
University of Virginia, Fall 2004

Ray Casting

```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;
}
```

Goal

- Must derive computer models for ...
  - Emission at light sources
  - Scattering at surfaces
  - Reception at the camera

• Desirable features ...
  - Concise
  - Efficient to compute
  - "Accurate"

Illumination

• How do we compute radiance for a sample ray?

```java
image[i][j] = GetColor(scene, ray, hit);
```

Angel Figure 6.2
Overview

• Direct (Local) Illumination
  ◦ Emission at light sources
  ◦ Scattering at surfaces

• Global illumination
  ◦ Shadows
  ◦ Refractions
  ◦ Inter-object reflections

Direct Illumination

Modeling Light Sources

• $l(x,y,z,\theta,\phi,\lambda)$ ...
  ◦ describes the intensity of energy,
  ◦ leaving a light source $L_0$, ...
  ◦ arriving at location $(x,y,z)$, ...
  ◦ from direction $(\theta,\phi)$, ...
  ◦ with wavelength $\lambda$

Empirical Models

• Ideally measure irradiant energy for "all" situations
  ◦ Too much storage
  ◦ Difficult in practice

OpenGL Light Source Models

• Simple mathematical models:
  ◦ Point light
  ◦ Directional light
  ◦ Spot light

Point Light Source

• Models omni-directional point source
  ◦ intensity ($I_0$),
  ◦ position $(px, py, pz)$,
  ◦ factors ($k_c, k_l, k_q$) for attenuation with distance ($d$)

$$I_L = \frac{I_0}{k_c + k_l d + k_q d^2}$$

Light

Directional Light Source

• Models point light source at infinity
  ◦ intensity ($I_0$),
  ◦ direction $(dx, dy, dz)$

No attenuation with distance

$I_L = I_0$
Spot Light Source

- Models point light source with direction
  - intensity ($I_0$),
  - position (px, py, pz),
  - direction (dx, dy, dz)
  - attenuation

\[ I_L = \frac{I_0 (D \cdot L)}{k_L + k_d d + k_a d^2} \]

Overview

- Direct Illumination
  - Emission at light sources
  - Scattering at surfaces

- Global illumination
  - Shadows
  - Refractions
  - Inter-object reflections

Modeling Surface Reflectance

- \( R_s(\theta, \phi; \gamma, \psi, \lambda) \) ...
  - describes the amount of incident energy,
  - arriving from direction (\( \theta, \phi \)), ...
  - leaving in direction (\( \gamma, \psi \)), ...
  - with wavelength \( \lambda \).

\[ R_s(\theta, \phi; \gamma, \psi, \lambda) \]

Empirical Models

- Ideally measure radiant energy for “all” combinations of incident angles
  - Too much storage
  - Difficult in practice

OpenGL Reflectance Model

- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - “ambient”

Based on model proposed by Phong

OpenGL Reflectance Model

- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - “ambient”

Based on model proposed by Phong
Diffuse Reflection

- Assume surface reflects equally in all directions
  - Examples: chalk, clay

How much light is reflected?
- Depends on angle of incident light

\[ dL = dA \cos \theta \]

OpenGL Reflectance Model

- Simple analytic model:
  - diffuse reflection
  - specular reflection
  - emission
  - “ambient”

Diffuse Reflection

Specular Reflection

- Reflection is strongest near mirror angle
  - Examples: mirrors, metals
Geometry of Reflection

\[ \theta_i = \theta_r \]

Specular Reflection

How much light is seen?

Depends on:
- angle of incident light
- angle to viewer

Specular Reflection

\[ R_s(I) = \cos(\theta_i)N \]

OpenGL Reflectance Model

- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - "ambient"

\[ I_s = K_s(V \cdot R)^* I_L \]
Emission

- Represents light emanating directly from polygon

Emission \neq 0

OpenGL Reflectance Model

- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - "ambient"

Ambient Term

- Represents reflection of all indirect illumination

This is a total hack (avoids complexity of global illumination)!

OpenGL Reflectance Model

- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - "ambient"

OpenGL Reflectance Model

- Sum diffuse, specular, emission, and ambient

Leonard McMillan, MIT
Surface Illumination Calculation

- Single light source:

\[ I = I_e + K_e I_a + K_{sh}(N \cdot L) I_e + K_s (V \cdot R)^\tau I_e \]

- Multiple light sources:

\[ I = I_e + K_e I_a + \sum_{i} (K_{sh}(N \cdot L) I_i + K_s (V \cdot R)^\tau I_i) \]

Overview

- Direct illumination
  - Emission at light sources
  - Scattering at surfaces

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

Global Illumination

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
  - \( 0 < S_i < 1 \) → soft shadows (hack)

\[ I = I_e + K_e I_a + \sum_{i} (K_{sh}(N \cdot L) + K_s (V \cdot R)^\tau) S_i I_e \]

Ray Casting (last lecture)

- Trace primary rays from camera
  - Direct illumination from unblocked lights only

\[ I = I_e + K_e I_a + \sum_{i} (K_{sh}(N \cdot L) + K_s (V \cdot R)^\tau) S_i I_e \]
Recursive Ray Tracing

- Also trace secondary rays from hit surfaces
  - Global illumination from mirror reflection and transparency

\[ I = I_e + K_d I_d + \sum_i (K_d (N \cdot L) + K_r (V \cdot R) S_i I_s) - K_s I_s + K_t I_t \]

Mirror reflections

- Trace secondary ray in mirror direction
  - Evaluate radiance along secondary ray and include it into illumination model

\[ I = I_e + K_d I_d + \sum_i (K_d (N \cdot L) + K_r (V \cdot R) S_i I_s) - K_s I_s + K_t I_t \]

Transparency

- Trace secondary ray in direction of refraction
  - Evaluate radiance along secondary ray and include it into illumination model

\[ I = I_e + K_d I_d + \sum_i (K_d (N \cdot L) + K_r (V \cdot R) S_i I_s) - K_s I_s + K_t I_t \]

Transparency

- Transparency coefficient is fraction transmitted
  - \( K_t = 1 \) for translucent object, \( K_t = 0 \) for opaque
  - \( 0 < K_t < 1 \) for object that is semi-translucent

Refractive Transparency

- For thin surfaces, can ignore change in direction
  - Assume light travels straight through surface

\[ T = (\eta_i \cos \Theta_i - \cos \Theta_r) N - \frac{\eta_i L}{\eta_r} \]

For solid objects, apply Snell’s law:

\[ \eta_i \sin \Theta_i = \eta_r \sin \Theta_r \]
Recursive Ray Tracing

- Ray tree represents illumination computation

\[
I = I_e + K_d I_d + \sum (K_o (N \cdot L) + K_v (V \cdot R)) S_f I_f + K_s I_s + K_a I_a
\]

Ray tree

Ray traced through scene

Recursive Ray Tracing

- Remember our ray caster?

```java
Image RayTrace(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;
}
```

Recursive Ray Tracing

- Need a new, recursive function "EvaluateRayTree"

```java
Color EvaluateRayTree(Scene scene, Ray ray, HitInformation hit) {
    Color contribution = black;
    for each light L:
        Ray shadow = Reflect(ray, hit.normal);
        contribution += Ks*EvaluateRayTree(scene, shadow, &hit);
        contribution += SpecularContribution(L, ray, hit);
        contribution += Ambient(); // hack-o-rama
        contribution += Emission(hit); // for area light sources only
    }
}
```

Recursive Ray Tracing

- Need a new, recursive function "EvaluateRayTree"

```java
Color ApplyLightingModel(Scene scene, Ray ray, HitInformation hit) {
    boolean hit_something;
    HitInformation hit; // structure containing hit point, normal, etc
    hit_something = FindIntersection( scene, ray, &hit );
    if (hit_something) {
        return ApplyLightingModel(ray, hit);
    } else {
        return BackgroundColor;
    }
}
```
Recursive Ray Tracing

```cpp
Color ApplyLightingModel(Scene scene, Ray ray, HitInformation hit)
{
    Color contribution = black;
    for each light L:
        Ray shadow(hit.pos, L.pos - hit.pos);
        HitInformation shadow_hit;
        bool blocked = FindIntersection(scene, shadow, &shadow_hit);
        if blocked && shadow_hit.t < Distance(L.pos, hit.pos):
            continue; // we’re in shadow, on to the next light;
        contribution += DiffuseContribution(L, hit);
        contribution += SpecularContribution(L, ray, hit);
        Ray mirror = Reflect(ray, hit.normal);
        contribution += K_r * EvaluateRayTree(scene, mirror);
        Ray glass = Refract(ray, hit.normal);
        contribution += K_t * EvaluateRayTree(scene, glass);
        contribution += Ambient(); // hack-o-rama
        contribution += Emission(hit); // for area light sources only
}
```

Precision

- Floating point calculations are imprecise!
- Often, a ray’s origin is supposed to be on a surface, but this might happen:

![Image](...)

- Typical hack is to only allow t values above some small threshold, like .0000001

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

Illumination Terminology

- Radiant power (flux) \( F \)
  - Rate at which light energy is transmitted (in Watts).
- Radiant Intensity \( I \)
  - Power radiated onto a unit solid angle in direction (in Watts/sr)
    - e.g.: energy distribution of a light source (inverse square law)
- Radiance \( L \)
  - Radiant intensity per unit projected surface area (in Watts/m²/sr)
    - e.g.: light carried by a single ray (no inverse square law)
- Irradiance \( E \)
  - Incident flux density on a locally planar area (in Watts/m²)
    - e.g.: light hitting a surface along a
- Radiosity \( B \)
  - Exitant flux density from a locally planar area (in Watts/m²)