Acceleration and Illumination

Michael Kazhdan

(600.357 / 600.457)

HB Ch. 14.1, 14.2
FvDFH 16.1, 16.2
Ray Casting

• Simple implementation:

```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(hit);
        }
    }
    return image;
}
```
Intersection Testing

Accelerated techniques try to leverage:

- **Grouping**: To efficiently discard groups of primitives that are guaranteed to be missed by the ray.
- **Ordering**: To test nearer intersections first and allow for early termination if there is a hit.
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
Binary Space Partition (BSP) Tree

• Recursively partition space by planes
  ◦ Generate a tree structure where the leaves store the shapes.
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Generate a tree structure where the leaves store the shapes.
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Generate a tree structure where the leaves store the shapes.

```
1
2  3
```

```
1  2  3
```

```
A  B  C  D  E  F
```
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Generate a tree structure where the leaves store the shapes.
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Generate a tree structure where the leaves store the shapes.
Binary Space Partition (BSP) Tree

- Example: Point Intersection
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on

Diagram:
- Nodes 1, 2, 3, 4, 5
- Shapes A, B, C, D, E, F
- Point $p$
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on
    - Left of 1 (root) → 2
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on
    - Left of 2 → 4
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on
    - Right of 4 → Test B
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on
    » Missed B. No intersection!
Binary Space Partition (BSP) Tree

- **Example: Ray Intersection 1**
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 1
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    - Test half to the left of 1
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 1
  ◦ Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Test half to the right of 2
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 1
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    - Intersection with C. Done!
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    - Test half to the left of 1
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Test half to the right of 2
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 2
  ◦ Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Missed C. Recurse!
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Test half to left of 2
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    - Test half to left of 4
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 2
  ◦ Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Missed A. Recurse!
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 2
  ◦ Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something: » No half to right of 4.
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    - Test half to right of 1
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    - Test half to left of 3
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 2
  ◦ Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Intersection with D. Done!
RayTreeIntersect(Ray ray, Node node, double min, double max)
{
    if (Node is a leaf) return intersection of closest primitive in cell, or NULL if none
    else
        // Find splitting point
        dist = distance along the ray to split plane of node

        // Find near and far children
        near_child = child of node that contains the origin of Ray
        far_child = other child of node

        // Recurse down near child first
        if( dist>min )
            {
                isect = RayTreeIntersect(ray, near_child, min, max)
                if( isect ) return isect    // If there’s a hit, we are done
            }

        // If there’s no hit, test the far child
        if( dist<max ) return RayTreeIntersect(ray, far_child, min, max)
}
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;
}
Ray Casting

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;
}
Illumination

• How do we compute radiance for a sample ray?

\[
image[i][j] = \text{GetColor( scene, ray, hit )};
\]
Goal

• Must derive models for ...
  ◦ Emission at light sources
  ◦ Direct light on surface points
  ◦ Scattering at surfaces
  ◦ Reception at the camera

• Desirable features …
  ◦ Concise
  ◦ Efficient to compute
  ◦ “Accurate”
Overview

• Direct Illumination
  ◦ Emission at a light source
  ◦ Reflection off the surface

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

• Direct Illumination
  ◦ Emission at light sources
  ◦ Reflection off the surface

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

Lambertian Shading
Overview

• Direct Illumination
  ◦ Emission at light sources
  ◦ Reflection off the surface

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

Phong Shading
Overview

• Direct Illumination
  ◦ Emission at light sources
  ◦ Reflection off the surface

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

Shadow Computation
Overview

• Direct Illumination
  ◦ Emission at light sources
  ◦ Reflection off the surface

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

Reflective Bouncing
Overview

• Direct Illumination
  ◦ Emission at light sources
  ◦ Reflection off the surface

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

Refractive Bouncing
Overview

• Direct Illumination
  ◦ Emission at light sources
  ◦ Reflection off the surface

• Global illumination
  ◦ Shadows
  ◦ Inter-object reflections
  ◦ Transmissions
Modeling Light Sources

- $I_L(x, y, z, \theta, \phi, \lambda)$
  - describes the intensity of energy ($I$),
  - leaving a light source ($L$),
  - at a particular angle ($\theta, \phi$)
  - arriving at a location ($x, y, z$),
  - with a particular wavelength ($\lambda$)
Empirical Models

- Ideally measure irradiant energy for “all” situations
  - Too much storage
  - Difficult in practice
Simplified Light Source Models

- Simple mathematical models:
  - Point light
  - Directional light
  - Spot light
Point Light Source

- Models omni-directional point source
  - intensity $I$,
  - position $p = (p_x, p_y, p_z)$,
  - factors $(k_c, k_l, k_q)$ for attenuation with distance ($\delta$)

$$I_L = \frac{I}{k_c + k_l \cdot \delta + k_q \cdot \delta^2}$$
Directional Light Source

- Models point light source at infinity
  - intensity $I$,
  - direction $\hat{d} = (d_x, d_y, d_z)$

No attenuation with distance

$$k_c = 1, k_l = k_q = 0$$
Spot Light Source

- Models point light source with direction
  - intensity $I$,
  - position $p = (p_x, p_y, p_z)$,
  - attenuation $(k_c, k_l, k_q)$
  - direction $\vec{d} = (d_x, d_y, d_z)$
  - cut-off and drop-off $(\gamma, \alpha)$

How can we modify point light to decrease as $\gamma$ increases?

$$I_L = \frac{I}{k_c + k_l \cdot \delta + k_q \cdot \delta^2}$$
Spot Light Source

- Models point light source with direction
  - intensity $I$,
  - position $p = (p_x, p_y, p_z)$,
  - attenuation $(k_c, k_l, k_q)$
  - direction $\hat{d} = (d_x, d_y, d_z)$
  - cut-off and drop-off $(\gamma, \alpha)$

\[
I_L = \begin{cases} 
I \cdot \frac{\langle \hat{d}, \hat{v} \rangle^\alpha}{k_c + k_l \cdot \delta + k_q \cdot \delta^2} & \text{if } \langle \hat{d}, \hat{v} \rangle > \cos \gamma \\
0 & \text{otherwise}
\end{cases}
\]
Overview

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

• Global illumination
  ◦ Shadows
  ◦ Transmissions
  ◦ Inter-object reflections
Modeling Surface Reflectance

- \( R_S(\theta, \phi, \lambda, \gamma, \psi) \)
  - describes the fraction of incident energy (\( R \)),
  - at the surface (\( S \)),
  - arriving from direction (\( \theta, \phi \)),
  - with wavelength (\( \lambda \)),
  - leaving in direction (\( \gamma, \psi \)),

\( \lambda \)

\( (\theta, \phi) \)

\( (\gamma, \psi) \)

Surface
Empirical Models

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

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

Based on model proposed by Phong
Simple 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
Diffuse Reflection

• How much light is reflected?
  ◦ Depends on angle of incident light
  ◦ aka “Lambertian”
Diffuse Reflection

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

\[ dL = dA \cdot \cos \theta \]
Diffuse Reflection

- Lambertian model
  - cosine law: \( \cos \theta = \langle \vec{N}, \vec{L} \rangle \)
  - \( K_D \) is surface property
  - \( I_L \) is incoming light

\[
I_D = K_D \cdot \langle \vec{N}, \vec{L} \rangle \cdot I_L
\]
Diffuse Reflection

- Note that lights and surface properties have R, G, and B components!
  - So amount of red light reflected is not necessarily equal to amount of green light, etc.
  - You will need to run calculation below on EACH color channel
  - This holds true for all lighting calculations

\[ I_{D}^{\text{Red}} = K_{D}^{\text{Red}} \cdot \langle \hat{N}, \hat{L} \rangle \cdot I_{L}^{\text{Red}} \]
Diffuse Reflection

• Assume surface reflects equally in all directions
  ◦ Examples: chalk, clay
Simple Reflectance Model

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

- Reflection is strongest near mirror angle
  - Examples: metals, shiny apples
Specular Reflection

How much light is seen?

Depends on how well the:

- reflected direction, and
- direction to the viewer

line up.
Specular Reflection

- Phong Model
  - $\cos^n \alpha$

This is a physically-motivated hack!

$$I_S = K_S \cdot \langle \hat{V}, \hat{R} \rangle^n \cdot I_L$$
Specular Reflection

• Reflection is strongest near mirror angle
  ◦ Examples: metals, shiny apples
Simple Reflectance Model

• Simple analytic model:
  ◦ diffuse reflection +
  ◦ specular reflection +
  ◦ emission +
  ◦ “ambient”
Emission

Represents light emanating directly from a surface that cannot be described by the three light sources

\[ \text{Emission} \neq 0 \]
Emission

\[ I_E \neq I_E \]
Simple 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)!
Ambient Term

- Represents reflection of all indirect illumination

\[ I_A = K_A \cdot I_L^A \]
Simple Reflectance Model

• Simple analytic model:
  ◦ diffuse reflection +
  ◦ specular reflection +
  ◦ emission +
  ◦ “ambient”
Simple Reflectance Model

• Simple analytic model:
  ◦ diffuse reflection +
  ◦ specular reflection +
  ◦ emission +
  ◦ “ambient”
Surface Illumination Calculation

- Single light source:

\[
I = I_E + K_A \cdot I_L^A + K_D \cdot \langle \vec{N}, \vec{L} \rangle \cdot I_L + K_S \cdot \langle \vec{V}, \vec{R} \rangle^n \cdot I_L
\]
Surface Illumination Calculation

- Single light source:

\[ I = I_E + \sum_L (K_A \cdot I_L^A + K_D \cdot \langle \vec{N}, \vec{L} \rangle \cdot I_L + K_S \cdot \langle \vec{V}, \vec{R} \rangle^n \cdot I_L) \]