3D Rendering and Ray Casting

Michael Kazhdan

(601.457/657)

HB Ch. 13.7, 14.6
FvDFH 15.5, 15.10
Rendering

- Generate an image from geometric primitives
What issues must be addressed by a 3D rendering system?
Overview

• 3D scene representation
• 3D viewer representation
• What do we see?
• How does it look?
Overview

• 3D scene representation
• 3D viewer representation
• What do we see?
• How does it look?

How is the 3D scene described in a computer?
3D Scene Representation

- Scene is usually approximated by 3D primitives
  - Point
  - Line segment
  - Polygon
  - Polyhedron
  - Curved surface
  - Solid object
  - etc.
3D Point

- Specifies a location

Origin
3D Point

• Specifies a location
  ◦ Represented by three coordinates
  ◦ Infinitely small

```
struct Point3D {
    float x, y, z;
};
```

\( (x, y, z) \)
3D Vector

• Specifies a direction and a magnitude
3D Vector

- Specifies a direction and a magnitude
  - Represented by three coordinates
  - Magnitude $\|\vec{v}\| = \sqrt{dx^2 + dy^2 + dz^2}$
  - Has no location

```c
struct Vector3D {
    float dx, dy, dz;
};
```

$\vec{v} = (dx, dy, dz)$
3D Vector

• Specifies a direction and a magnitude
  ◦ Represented by three coordinates
  ◦ Magnitude \( \|\mathbf{v}\| = \sqrt{dx^2 + dy^2 + dz^2} \)
  ◦ Has no location

• Dot product of two 3D vectors
  ◦ \( \langle \mathbf{v}_1, \mathbf{v}_2 \rangle = dx_1 \cdot dx_2 + dy_1 \cdot dy_2 + dz_1 \cdot dz_2 \)
  ◦ \( \langle \mathbf{v}_1, \mathbf{v}_2 \rangle = \|\mathbf{v}_1\| \cdot \|\mathbf{v}_2\| \cdot \cos \theta \)

• Cross product of two 3D vectors
  ◦ \( \mathbf{v}_1 \times \mathbf{v}_2 = \text{Vector normal to plane } \mathbf{v}_1, \mathbf{v}_2 \)
  ◦ \( \|\mathbf{v}_1 \times \mathbf{v}_2\| = \|\mathbf{v}_1\| \cdot \|\mathbf{v}_2\| \cdot \sin \theta \)
Cross Product: Review

• Let $\mathbf{v}_1 = \mathbf{v}_2 \times \mathbf{v}_3$:
  - $dx_1 = dy_2 \cdot dz_3 - dz_2 \cdot dy_3$
  - $dy_1 = dz_2 \cdot dx_3 - dx_2 \cdot dz_3$
  - $dz_1 = dx_2 \cdot dy_3 - dy_2 \cdot dx_3$

• $\mathbf{v} \times \mathbf{w} = -\mathbf{w} \times \mathbf{v}$ (remember “right-hand” rule)

• We can show:
  - $\mathbf{v} \times \mathbf{w} = ||\mathbf{v}|| \cdot ||\mathbf{w}|| \cdot \sin \theta \cdot \mathbf{n}$, where $\mathbf{n}$ is unit vector normal to $\mathbf{v}$ and $\mathbf{w}$
  - $\mathbf{v} \times \mathbf{v} = 0$
3D Line Segment

- Linear path between two points
3D Line Segment

- Use a linear combination of two points
  - Parametric representation:
    \[ p(t) = p_1 + t \cdot (p_2 - p_1), \quad (0 \leq t \leq 1) \]

```c
struct Segment3D {
    Point3D p1, p2;
};
```
3D Ray

• Line segment with one endpoint at infinity
  ◦ Parametric representation:
    \[ p(t) = p_1 + t \cdot \vec{v}, \quad (0 \leq t < \infty) \]

```c
struct Ray3D
{
    Point3D p1;
    Vector3D v;
};
```
3D Line

- Line segment with both endpoints at infinity
  - Parametric representation:
    \[ p(t) = p_1 + t \cdot \vec{v}, \quad (-\infty < t < \infty) \]

```c
struct Line3D {
    Point3D p1;
    Vector3D v;
};
```
3D Plane

• A linear combination of three points
3D Plane

- A linear combination of three points
  - Implicit representation:
    - $\Phi(p) = ax + by + cz - d = 0$
    - $\Phi(p) = \langle p, \vec{n} \rangle - d = 0$, or
    ```
    struct Plane3D {
      Vector3D n;
      float d;
    };
    ``
  - $\vec{n}$ is the plane normal
    - (May be) unit-length vector
    - Perpendicular to plane
  - $d$ is the signed (weighted) distance of the plane from the origin.
3D Polygon

- Area “inside” a sequence of coplanar points
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting

```c
struct Polygon3D {
    Point3D *points;
    int npoints;
};
```

Points are in counter-clockwise order

- Holes (use > 1 polygon struct)
3D Sphere

- All points at distance $r$ from center point $c = (c_x, c_y, c_z)$
  - Implicit representation:
    » $\Phi(p) = ||p - c||^2 - r^2 = 0$
  - Parametric representation:
    » $x(\phi, \theta) = r \cdot \cos \phi \cdot \sin \theta + c_x$
    » $y(\phi, \theta) = r \cdot \cos \phi \cdot \sin \theta + c_y$
    » $z(\theta, \phi) = r \cdot \sin \phi + c_z$

```
struct Sphere3D {
    Point3D center;
    float radius;
};
```
Other 3D primitives

- Cone
- Cylinder
- Ellipsoid
- Box
- Etc.
3D Geometric Primitives

• More detail on 3D modeling later in course
  ○ Point
  ○ Line segment
  ○ Polygon
  ○ Polyhedron
  ○ Curved surface
  ○ Solid object
  ○ etc.
Overview

- 3D scene representation
- 3D viewer representation
- What do we see?
- How does it look?

How is the viewing device described in a computer?
Camera Models

• The most common model is pin-hole camera
  ○ All captured light rays arrive along paths toward focal point without lens distortion (everything is in focus)

Other models consider ...
  Depth of field
  Motion blur
  Lens distortion
Camera Parameters

• What are the parameters of a camera?
Camera Parameters

• Position
  ◦ Eye position: \texttt{Point3D eye}

• Orientation
  ◦ View direction: \texttt{Vector3D view}
  ◦ Up direction: \texttt{Vector3D up}

• Aperture
  ◦ Field of view: float \texttt{xFov}, \texttt{yFov}
  ◦ Film plane
  ◦ (View plane normal)
Other Models: Depth of Field

Close Focused

Distance Focused

P. Haeberli
Other Models: Motion Blur

• Mimics effect of open camera shutter
• Gives perceptual effect of high-speed motion
• Generally involves temporal super-sampling
Traditional Pinhole Camera

- The film sits behind the pinhole of the camera.
- Rays come in from the outside, pass through the pinhole, and hit the film plane.
Traditional Pinhole Camera

- The film sits behind the pinhole of the camera.
- Rays come in from the outside, pass through the pinhole, and hit the film plane.

Photograph is upside down
Virtual Camera

- The film sits in front of the pinhole of the camera.
- Rays come in from the outside, pass through the film plane, and hit the pinhole.
Virtual Camera

- The film sits in front of the pinhole of the camera.
- Rays come in from the outside, pass through the film plane, and hit the pinhole.

Photograph is right side up
Overview

• 3D scene representation
• 3D viewer representation
• Ray Casting
  ○ Where are we looking?
  ○ What do we see?
  ○ How does it look?
Ray Casting

- For each sample …
  - **Where**: Construct ray from eye through view plane
  - **What**: Find first surface intersected by ray through pixel
  - **How**: Compute color sample based on surface radiance
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;
}
```
Ray Casting

Where?

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;
Constructing a Ray Through a Pixel

Up direction

View Plane

$p_0$

$towards$

$p[i][j]$
The ray has to originate at \( p_0 \) (the position of the camera). So the equation for the ray is:

\[
\text{Ray}(t) = p_0 + t \cdot \vec{v}
\]
Constructing a Ray Through a Pixel

If the ray passes through the point \( p[i][j] \), then the solution for \( \vec{v} \) is:

\[
\vec{v} = \frac{p[i][j] - p_0}{\|p[i][j] - p_0\|}
\]
If \( p[i][j] \) represents the \((i, j)\)-th pixel of the image, what is its position?
Constructing Ray Through a Pixel

- 2D Example: Side view of camera at \( p_0 \)
  - Where is the \( i \)-th pixel, \( p[i] \)? (\( i \in [0, \text{height}) \))

\[ \theta = \text{frustum half-angle (given), or field of view} \]
\[ d = \text{distance to view plane (arbitrary = you pick)} \]
Constructing Ray Through a Pixel

• 2D Example: Side view of camera at $p_0$
  ◦ Where is the $i$-th pixel, $p[i]$? ($i \in [0, \text{height}]$)

$\theta =$ frustum half-angle (given), or field of view

$d =$ distance to view plane (arbitrary = you pick)

$p_1 = p_0 + d \cdot \text{towards} - d \cdot \tan \theta \cdot \text{up}$

$p_2 = p_0 + d \cdot \text{towards} + d \cdot \tan \theta \cdot \text{up}$
Constructing Ray Through a Pixel

- 2D Example: Side view of camera at $p_0$
  - Where is the $i$-th pixel, $p[i]$? ($i \in [0, \text{height}]$)

\[ \theta = \text{frustum half-angle (given), or field of view} \]
\[ d = \text{distance to view plane (arbitrary = you pick)} \]

\[ p_1 = p_0 + d \cdot \text{towards} - d \cdot \tan \theta \cdot \text{up} \]
\[ p_2 = p_0 + d \cdot \text{towards} + d \cdot \tan \theta \cdot \text{up} \]

\[ p[i] = p_1 + \left( \frac{i + 0.5}{\text{height}} \right) \cdot (p_2 - p_1) \]
\[ = p_1 + \left( \frac{i + 0.5}{\text{height}} \right) \cdot 2 \cdot d \cdot \tan \theta \cdot \text{up} \]
Constructing Ray Through a Pixel

• 2D Example:

The ray passing through the $i$-th pixel is defined by:

$$\text{Ray}(t) = p_0 + t \cdot \hat{v}$$

○ $p_0$: camera position
○ $\hat{v}$: direction to the $i$-th pixel:

$$\hat{v} = \frac{p[i] - p_0}{\|p[i] - p_0\|}$$

○ $p[i]$: $i$-th pixel location:

$$p[i] = p_1 + \left(\frac{i + 0.5}{\text{height}}\right) \cdot (p_2 - p_1)$$

○ $p_1$ and $p_2$ are the endpoints of the view plane:

$$p_1 = p_0 + d \cdot \text{towards} - d \cdot \tan \theta \cdot \text{up}$$
$$p_2 = p_0 + d \cdot \text{towards} + d \cdot \tan \theta \cdot \text{up}$$
Constructing Ray Through a Pixel

- Figuring out how to do this in 3D is assignment 2
Ray Casting

Where?

```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;
}
```
Ray Casting

What?

```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;
}
```
Ray-Scene Intersection

- Intersections with geometric primitives
  - Sphere
  - Triangle
Ray-Sphere Intersection

Ray: \( p(t) = p_0 + t \cdot \hat{v}, \quad (0 \leq t < \infty) \)

Sphere: \( \Phi(p) = \|p - c\|^2 - r^2 = 0 \)
Ray-Sphere Intersection

Ray: \( p(t) = p_0 + t \cdot \vec{v}, \quad (0 \leq t < \infty) \)

Sphere: \( \Phi(p) = ||p - c||^2 - r^2 = 0 \)

Substituting for \( p(t) \), we get:
\( \Phi(t) = ||p_0 - t \cdot \vec{v} - c||^2 - r^2 = 0 \)
Ray-Sphere Intersection

Ray: \( p(t) = p_0 + t \cdot \vec{v}, \ (0 \leq t < \infty) \)
Sphere: \( \Phi(p) = \|p - c\|^2 - r^2 = 0 \)

Substituting for \( p(t) \), we get:
\( \Phi(t) = \|p_0 - t \cdot \vec{v} - c\|^2 - r^2 = 0 \)

Solve quadratic equation:
\[ a \cdot t^2 + b \cdot t + c = 0 \]
where:
\[ a = 1 \]
\[ b = 2 \langle \vec{v}, p_0 - c \rangle \]
\[ c = \|p_0 - c\|^2 - r^2 \]
Ray-Sphere Intersection

Ray: \( p(t) = p_0 + t \cdot \vec{v}, \ (0 \leq t < \infty) \)
Sphere: \( \Phi(p) = \|p - c\|^2 - r^2 = 0 \)

Substituting for \( p(t) \), we get:
\[
\Phi(t) = \|p_0 - t \cdot \vec{v} - c\|^2 - r^2 = 0
\]

Solve quadratic equation:
\[
a \cdot t^2 + b \cdot t + c = 0
\]
where:

generally, there are two solutions to the quadratic equation, giving rise to points \( p \) and \( p' \).
want to return the first positive hit.
Ray-Sphere Intersection

• Need normal vector at intersection for lighting calculations:

\[ \mathbf{n} = \frac{\mathbf{p} - \mathbf{c}}{\|\mathbf{p} - \mathbf{c}\|} \]
Ray-Scene Intersection

• Intersections with geometric primitives
  ◦ Sphere
  » Triangle
Ray-Triangle Intersection

• First, intersect ray with plane
• Then, check if point is inside triangle
Ray-Plane Intersection

Ray: \( p(t) = p_0 + t \cdot \vec{v}, \quad (0 \leq t < \infty) \)

Plane: \( \Phi(p) = \langle p, \vec{n} \rangle - d = 0 \)

Substituting for \( P \), we get:
\[
\Phi(t) = \langle p_0 + t \cdot \vec{v}, \vec{n} \rangle - d = 0
\]

Solution:
\[
t = -\frac{\langle p_0, \vec{n} \rangle - d}{\langle \vec{v}, \vec{n} \rangle}
\]
Ray-Triangle Intersection I

• Check if point is inside triangle algebraically:
  - Generate triangles by connecting the ray source to each edge
  - Check if the point of intersection is above each of these triangles

For each side of triangle
\[
\vec{v}_1 = T_1 - p_0 \\
\vec{v}_2 = T_2 - p_0 \\
\vec{n}_1 = \vec{v}_2 \times \vec{v}_1
\]
if \( \langle p - p_0, \vec{n}_1 \rangle < 0 \)
return FALSE;
A point $p$ is inside the triangle iff. it can be expressed as the weighted average of the corners:

$$p = \alpha \cdot T_1 + \beta \cdot T_2 + \gamma \cdot T_3$$

where:

$$0 \leq \alpha, \beta, \gamma \leq 1$$

$$\alpha + \beta + \gamma = 1$$
Ray-Triangle Intersection II

- Check if point is inside triangle parametrically

Solve for $\alpha, \beta, \gamma$ such that:

$$p = \alpha \cdot T_1 + \beta \cdot T_2 + \gamma \cdot T_3$$

And

$$\alpha + \beta + \gamma = 1$$

Check if the point is in the triangle:

$$0 \leq \alpha, \beta, \gamma \leq 1$$