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

Geometric Primitives (3D)
Rendering

• Generate an image from geometric primitives
3D Rendering Example

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
  - Triangles
  - Polygon
  - Polyhedron
  - Curved surface
  - Solid object
  - etc.
3D Point

- Specifies a location
3D Point

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

```c
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 $||\vec{v}|| = \sqrt{dx^2 + dy^2 + dz^2}$
  - Has no location

- Dot product of two 3D vectors
  - $\langle \vec{v}_1, \vec{v}_2 \rangle = dx_1 \cdot dx_2 + dy_1 \cdot dy_2 + dz_1 \cdot dz_2$
  - $\langle \vec{v}_1, \vec{v}_2 \rangle = ||\vec{v}_1|| \cdot ||\vec{v}_2|| \cdot \cos \theta$

- Cross product of two 3D vectors
  - $\vec{v}_1 \times \vec{v}_2 =$ Vector normal to $v_1$ and $v_2$
  - $||\vec{v}_1 \times \vec{v}_2|| = ||\vec{v}_1|| \cdot ||\vec{v}_2|| \cdot \sin \theta$
Cross Product: Review

- Let \( \vec{v}_1 = \vec{v}_2 \times \vec{v}_3 \):
  - \( dx_1 = dy_2 \cdot dz_3 - dz_2 \cdot dy_3 \)
  - \( dy_1 = dz_2 \cdot dx_3 - dx_2 \cdot dz_3 \)
  - \( dz_1 = dx_2 \cdot dy_3 - dy_2 \cdot dx_3 \)

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

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

- Linear path between two points
3D Line Segment

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

```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:
    » \( \Phi(p) = ax + by + cz - d = 0 \)
    » \( \Phi(p) = \langle p, \vec{n} \rangle - d = 0 \)

\[
\text{struct Plane3D}
\{
    \text{Vector3D n;}
    \text{float d;}
\};
\]

○ \( \vec{n} \) 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

```
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:**
    
    \[
    \begin{align*}
    x(\phi, \theta) &= r \cdot \cos \phi \cdot \sin \theta + c_x \\
    y(\phi, \theta) &= r \cdot \cos \phi \cdot \sin \theta + c_y \\
    z(\theta, \phi) &= r \cdot \sin \phi + c_z
    \end{align*}
    \]

```c
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
  - Triangle
  - 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: `Point3D eye`

- Orientation
  - View direction: `Vector3D view`
  - Up direction: `Vector3D up`

- Aperture
  - Field of view angle: `float xFov, yFov`
  - Resolution of film plane: `int width, height`
  - Distance of film plane
  - (Orientation of film plane)
Other Models: Depth of Field

Close Focused

Distance Focused
Other Models: Motion Blur

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

Brostow & Essa
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?

```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;
}
```
Constructing a Ray Through a Pixel
The ray originates 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:

\[
\vec{v} = \frac{p[i][j] - p_0}{||p[i][j] - p_0||}
\]
Constructing a Ray Through a Pixel

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, $p[i]$? ($i \in [0, \text{height})$)

$\theta = \text{field of view angle (given)}$

$d = \text{distance to view plane (arbitrary = you pick)}$
Constructing Ray Through a Pixel

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

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

\[
p_1 = p_0 + d \cdot \text{towards} - d \cdot \tan \frac{\theta}{2} \cdot \text{up}
\]

\[
p_2 = p_0 + d \cdot \text{towards} + d \cdot \tan \frac{\theta}{2} \cdot \text{up}
\]
Constructing Ray Through a Pixel

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

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

\[
\begin{align*}
p_1 &= p_0 + d \cdot \text{towards} - d \cdot \tan \frac{\theta}{2} \cdot \text{up} \\
p_2 &= p_0 + d \cdot \text{towards} + d \cdot \tan \frac{\theta}{2} \cdot \text{up} \\
p[i] &= p_1 + \left(\frac{i + 0.5}{\text{height}}\right) \cdot (p_2 - p_1)
\end{align*}
\]
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:
  $$p_1 = p_0 + d \cdot \text{towards} - d \cdot \tan \theta/2 \cdot \text{up}$$
  $$p_2 = p_0 + d \cdot \text{towards} + d \cdot \tan \theta/2 \cdot \text{up}$$
Constructing Ray Through a Pixel

Figuring out how to do this in 3D is assignment 2
Constructing Ray Through a Pixel

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

Note:

- Given the vertical field of view angle, $\theta_v$
Constructing Ray Through a Pixel

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

Note:

- Given the vertical field of view angle, $\theta_v$
- And the aspect ratio, $ar = \frac{\text{height}}{\text{width}}$
Constructing Ray Through a Pixel

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

Note:

- Given the vertical field of view angle, $\theta_v$
- And the aspect ratio, $ar = \frac{height}{width}$

The horizontal field of view angle, $\theta_h$, satisfies:

$$\frac{\tan(\theta_v/2)}{\tan(\theta_h/2)} = ar$$
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;
}
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: \( p(t) = p_0 + t \cdot \vec{v}, \quad (0 \leq t < \infty) \)
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 \( p \), 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 \( p \), 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 two points of intersection, \( p \) and \( p' \). Want to return the first positive hit.
Ray-Sphere Intersection

- Need normal vector at intersection for lighting calculations:

\[ \vec{n} = \frac{\vec{p} - \vec{c}}{||\vec{p} - \vec{c}||} \]
Ray-Sphere Intersection

• More generally, if the shape is given as the set of points $p$ satisfying:
  \[ \Phi(p) = 0 \]
for some function $\Phi: \mathbb{R}^3 \rightarrow \mathbb{R}$, then the normal of the surface will be parallel to the gradient.
Ray-Scene Intersection

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

1. Intersect ray with plane
2. Check if the point is inside the triangle
Ray-Plane Intersection

Ray: \( p(t) = p_0 + t \cdot \vec{v}, \) \( 0 \leq t < \infty \)
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:
\[ t = -\frac{\langle p_0, \vec{n} \rangle - d}{\langle \vec{v}, \vec{n} \rangle} \]

Algebraic Method

What are the implications of \( \langle \vec{v}, \vec{n} \rangle = 0 \)?
Ray-Triangle Intersection I

- Check for point-triangle intersection algebraically:
  - Generate planes through the ray source and each edge
  - Check if the point of intersection is above each of these planes

For each edge:
\[ \mathbf{n}_i = (\mathbf{v}_{i+1} - \mathbf{p}_0) \times (\mathbf{v}_i - \mathbf{p}_0) \]
if \( \langle \mathbf{p} - \mathbf{p}_0, \mathbf{n}_i \rangle < 0 \) return FALSE;
Ray-Triangle Intersection II

• Check for point-triangle intersection parametrically

In general, given \( p \in \mathbb{R}^3 \) and given three points \( \{v_1, v_2, v_3\} \subset \mathbb{R}^3 \) (in general position) we can solve for \( \alpha, \beta, \gamma \in \mathbb{R} \) such that:

\[
 p = \alpha v_1 + \beta v_2 + \gamma v_3
\]

\( p \) is in the plane spanned by \( \{v_1, v_2, v_3\} \) iff.:

\[
 \alpha + \beta + \gamma = 1
\]

\( p \) is inside the triangle with vertices \( \{v_1, v_2, v_3\} \) iff.:

\[
 \alpha, \beta, \gamma \geq 0
\]
Ray-Triangle Intersection II

• Check for point-triangle intersection parametrically

In general, given \( p \in \mathbb{R}^3 \) and given three points \( \{v_1, v_2, v_3\} \subset \mathbb{R}^3 \) (in general position) we can solve for \( \alpha, \beta, \gamma \in \mathbb{R} \) such that:

\[
p = \alpha v_1 + \beta v_2 + \gamma v_3
\]

To get \( \alpha, \beta, \gamma \), solve the system:

\[
\begin{pmatrix}
v_1^x & v_2^x & v_3^x \\
v_1^y & v_2^y & v_3^y \\
v_1^z & v_2^z & v_3^z
\end{pmatrix}
\begin{pmatrix}
\alpha \\
\beta \\
\gamma
\end{pmatrix}
=
\begin{pmatrix}
p^x \\
p^y \\
p^z
\end{pmatrix}
\]
Ray-Triangle Intersection II

• Check for point-triangle intersection parametrically

In general, given \( p \in \mathbb{R}^3 \) and given three points \( \{v_1, v_2, v_3\} \subset \mathbb{R}^3 \) (in general position) we can solve for \( \alpha, \beta, \gamma \in \mathbb{R} \) such that:

\[
p = \alpha v_1 + \beta v_2 + \gamma v_3
\]

To get \( \alpha, \beta, \gamma \), solve the system:

\[
\begin{bmatrix}
  v_1^x & v_2^x & v_3^x \\
  v_1^y & v_2^y & v_3^y \\
  v_1^z & v_2^z & v_3^z
\end{bmatrix}
\begin{bmatrix}
  \alpha \\
  \beta \\
  \gamma
\end{bmatrix} =
\begin{bmatrix}
  p^x \\
  p^y \\
  p^z
\end{bmatrix}
\iff
\begin{bmatrix}
  \alpha \\
  \beta \\
  \gamma
\end{bmatrix} =
\begin{bmatrix}
  v_1^x & v_2^x & v_3^x \\
  v_1^y & v_2^y & v_3^y \\
  v_1^z & v_2^z & v_3^z
\end{bmatrix}^{-1}
\begin{bmatrix}
  p^x \\
  p^y \\
  p^z
\end{bmatrix}
\]
Ray-Triangle Intersection II

• Check for point-triangle intersection parametrically

In general, given $p \in \mathbb{R}^3$ and given three points $\{v_1, v_2, v_3\} \subset \mathbb{R}^3$ (in general position) we can solve for $\alpha, \beta, \gamma \in \mathbb{R}$ such that:

$$p = \alpha v_1 + \beta v_2 + \gamma v_3$$

To get $\alpha, \beta, \gamma$, solve the system:

$$\begin{pmatrix} v_1^x & v_2^x & v_3^x \\ v_1^y & v_2^y & v_3^y \\ v_1^z & v_2^z & v_3^z \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \iff \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} v_1^x & v_2^x & v_3^x \\ v_1^y & v_2^y & v_3^y \\ v_1^z & v_2^z & v_3^z \end{pmatrix}^{-1} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

This will fail if the vertices $\{v_1, v_2, v_3\}$ lie in a plane through the origin.
Ray-Triangle Intersection II

• Check for point-triangle intersection parametrically

Embrace the problem case by translating the whole system to the origin:

\[
\begin{pmatrix}
  v_1^x & v_2^x & v_3^x \\
  v_1^y & v_2^y & v_3^y \\
  v_1^z & v_2^z & v_3^z \\
\end{pmatrix} \begin{pmatrix}
  \alpha \\
  \beta \\
  \gamma \\
\end{pmatrix} = \begin{pmatrix}
  p^x \\
  p^y \\
  p^z \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
  0 & v_2^x - v_1^x & v_3^x - v_1^x \\
  0 & v_2^y - v_1^y & v_3^y - v_1^y \\
  0 & v_2^z - v_1^z & v_3^z - v_1^z \\
\end{pmatrix} \begin{pmatrix}
  \alpha \\
  \beta \\
  \gamma \\
\end{pmatrix} = \begin{pmatrix}
  p^x - v_1^x \\
  p^y - v_1^y \\
  p^z - v_1^z \\
\end{pmatrix}
\]
Ray-Triangle Intersection II

- Check for point-triangle intersection parametrically

Embrace the problem case by translating the whole system to the origin:

\[
\begin{pmatrix}
    v_1^x & v_2^x & v_3^x \\
    v_1^y & v_2^y & v_3^y \\
    v_1^z & v_2^z & v_3^z
\end{pmatrix}
\begin{pmatrix}
    \alpha \\
    \beta \\
    \gamma
\end{pmatrix} =
\begin{pmatrix}
    p^x \\
    p^y \\
    p^z
\end{pmatrix}
\]

\[
\begin{pmatrix}
    v_2^x - v_1^x & v_3^x - v_1^x \\
    v_2^y - v_1^y & v_3^y - v_1^y \\
    v_2^z - v_1^z & v_3^z - v_1^z
\end{pmatrix}
\begin{pmatrix}
    \beta \\
    \gamma
\end{pmatrix} =
\begin{pmatrix}
    p^x - v_1^x \\
    p^y - v_1^y \\
    p^z - v_1^z
\end{pmatrix}
\]
Ray-Triangle Intersection II

- Check for point-triangle intersection parametrically

Embrace the problem case by translating the whole system to the origin:

\[
\begin{align*}
\alpha & = \mathbf{p} - \mathbf{v}_1 \\
\beta & = \frac{\mathbf{p} - \mathbf{v}_1}{\mathbf{v}_2 - \mathbf{v}_1} \\
\gamma & = \frac{\mathbf{p} - \mathbf{v}_1}{\mathbf{v}_3 - \mathbf{v}_1}
\end{align*}
\]

\[
\begin{bmatrix}
\mathbf{v}_1^x & \mathbf{v}_2^x & \mathbf{v}_3^x \\
\mathbf{v}_1^y & \mathbf{v}_2^y & \mathbf{v}_3^y \\
\mathbf{v}_1^z & \mathbf{v}_2^z & \mathbf{v}_3^z
\end{bmatrix}
\begin{bmatrix}
\alpha \\
\beta \\
\gamma
\end{bmatrix}
=
\begin{bmatrix}
\mathbf{p}^x \\
\mathbf{p}^y \\
\mathbf{p}^z
\end{bmatrix}
\]

This is an over-constrained system!
In general, we can’t express a 3D point as the linear combination of two 3D points.

This is not the general case!
A solution exists since \( \mathbf{p} \) is in the plane spanned by \( \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\} \)

After solving for \( \beta \) and \( \gamma \), we can set:

\[
\alpha = 1 - \beta - \gamma
\]
Other Ray-Primitive Intersections

• Cone, cylinder, ellipsoid:
  ◦ Similar to sphere

• Box
  ◦ Intersect 3 front-facing planes, return closest

• Convex polygon
  » Find the intersection of the ray with the plane
  » Check that the intersection is above every triangle generated by the ray source and polygon edge.

• Concave polygon
  ◦ Same plane intersection
  ◦ More complex point-in-polygon test