3D Polygon Rendering Pipeline
Adam Finkelstein & Tim Weyrich
Princeton University
COS 426, Spring 2008

3D Rendering Scenarios
- Batch
  - One image generated with as much quality as possible for a particular set of rendering parameters
    - Take as much time as is needed (minutes)
    - Useful for photorealism, movies, etc.
- Interactive
  - Images generated in fraction of a second (<1/10) as user controls rendering parameters (e.g., camera)
    - Achieve highest quality possible in given time
    - Useful for visualization, games, etc.

3D Polygon Rendering
- Many applications use rendering of 3D polygons with direct illumination

Ray Casting Revisited
- For each sample …
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color of sample based on surface radiance

3D Polygon Rendering
- We can render polygons faster if we take advantage of spatial coherence

More efficient algorithms utilize spatial coherence!
This is a pipelined sequence of operations to draw a 3D primitive into a 2D image.

OpenGL executes steps of the 3D rendering pipeline for each polygon.
3D Rendering Pipeline (for direct illumination)

3D Primitives → Modeling Transformation → Lighting → Viewport Transformation → Projection Transformation → Clipping → Scan Conversion → Image

Transform into 3D world coordinate system
Illuminate according to lighting and reflectance
Transform into 3D camera coordinate system
Clip primitives outside camera’s view
Transform into 2D camera coordinate system
Transform into image coordinate system
3D Rendering Pipeline (for direct illumination)

- Transform into 3D world coordinate system
- Illuminate according to lighting and reflectance
- Transform into 3D camera coordinate system
- Transform into 2D camera coordinate system
- Clip primitives outside camera’s view
- Transform into image coordinate system
- Draw pixels (includes texturing, hidden surface, ...)

Transformations map points from one coordinate system to another

- p(x,y,z)
- Transformations map points from one coordinate system to another

Viewing Transformation

- Mapping from world to camera coordinates
  - Eye position maps to origin
  - Right vector maps to X axis
  - Up vector maps to Y axis
  - Back vector maps to Z axis

Camera Coordinates

- Canonical coordinate system
  - Convention is right-handed (looking down -z axis)
  - Convenient for projection, clipping, etc.

Camera up vector maps to Y axis
Camera back vector maps to Z axis (pointing out of page)
Finding the Viewing Transformation

• Trick: map from camera coordinates to world
  - Origin maps to eye position
  - Z axis maps to Back vector
  - Y axis maps to Up vector
  - X axis maps to Right vector

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} = \begin{bmatrix}
a & b & c & d \\
e & f & g & h \\
i & j & k & l \\
m & n & o & p
\end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]

• This matrix is \(T^{-1}\) so we invert it to get \(T\) ... easy!

Projection

• General definition:
  - Transform points in \(n\)-space to \(m\)-space \((m<n)\)

• In computer graphics:
  - Map 3D camera coordinates to 2D screen coordinates
Parallel Projection

- Center of projection is at infinity
  - Direction of projection (DOP) same for all points

Orthographic Projections

- DOP perpendicular to view plane

Oblique Projections

- DOP not perpendicular to view plane
  - Cavalier (DOP $\alpha = 45^\circ$)
  - Cabinet (DOP $\alpha = 63.4^\circ$)

Parallel Projection View Volume

- Parallelepiped View Volume

Parallel Projection Matrix

- General parallel projection transformation:

\[
\begin{bmatrix}
x'_p \\
y'_p \\
z'_p \\
w'_p
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & L \cos \phi & 0 \\
0 & 1 & L \sin \phi & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_p \\
y_p \\
z_p \\
w_p
\end{bmatrix}
\]

Taxonomy of Projections

- Planar geometric projections
- Perspective
- Parallel
- Orthographic
- Oblique
- One-point
- Two-point
- Three-point
- Top elevation
- Axonometric
- Cabinet
- Cavalier
- Isometric
- Other
Perspective Projection

- Map points onto "view plane" along "projectors" emanating from "center of projection" (COP).

[Diagram of Perspective Projection]

Angel Figure 5.9

Perspective Projection

- How many vanishing points?

- 3-Point Perspective
- 2-Point Perspective
- 1-Point Perspective

Angel Figure 5.10

Perspective Projection View Volume

- Compute 2D coordinates from 3D coordinates with similar triangles.

H&B Figure 12.30

Perspective Projection

- Compute 2D coordinates from 3D coordinates with similar triangles.

- What are the coordinates of the point resulting from projection of (x,y,z) onto the view plane?

Perspective Projection Matrix

- 4x4 matrix representation?

\[
\begin{bmatrix}
    x_s \\
    y_s \\
    z_s \\
    w_s
\end{bmatrix} = \begin{bmatrix}
    ? & ? & ? & 1
\end{bmatrix} \begin{bmatrix}
    x_c \\
    y_c \\
    z_c \\
    1
\end{bmatrix}
\]

[Matrix representation diagram]
Perspective Projection Matrix

- 4x4 matrix representation?

\[
\begin{align*}
    x'_c &= x_c D / z_c \\
    y'_c &= y_c D / z_c \\
    z'_c &= D \\
    w'_c &= z_c / D \\
\end{align*}
\]

\[
\begin{bmatrix}
    x'_c \\
    y'_c \\
    z'_c \\
    w'_c \\
\end{bmatrix} = \begin{bmatrix}
    x_c/D \\
    y_c/D \\
    D \\
    z_c/D \\
\end{bmatrix}
\]

Perspective vs. Parallel

- Perspective projection
  + Size varies inversely with distance - looks realistic
  - Distance and angles are not (in general) preserved
  - Parallel lines do not (in general) remain parallel

- Parallel projection
  + Good for exact measurements
  + Parallel lines remain parallel
  - Angles are not (in general) preserved
  - Less realistically looking

Taxonomy of Projections

Classical Projections

Viewing Transformations Summary

- Camera transformation
  - Map 3D world coordinates to 3D camera coordinates
  - Matrix has camera vectors as rows

- Projection transformation
  - Map 3D camera coordinates to 2D screen coordinates
  - Two types of projections:
    - Parallel
    - Perspective
### 3D Rendering Pipeline (for direct illumination)

- **Modeling Transformation**
- **Projection Transformation**
- **Clipping**
- **Viewing Transformation**
- **Scan Conversion**
- **Image**

### 2D Rendering Pipeline

- **Viewport Transformation**
- **Clip Conversion**
- **Image**

### Clipping

- Avoid drawing parts of primitives outside window
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window

---

- **3D Primitives**
- **2D Primitives**
- **Clipping**
- **Viewport Transformation**
- **Scan Conversion**
- **Image**

- **3D Modeling Coordinates**
- **3D World Coordinates**
- **3D Camera Coordinates**
- **2D Screen Coordinates**
Clipping

- Avoid drawing parts of primitives outside window
  - Points
  - Lines
  - Polygons
  - Circles
  - etc.

Point Clipping

- Is point \((x,y)\) inside the clip window?

\[
\text{inside} = \begin{cases} 
(x \geq wx1) & \text{&&} \\
(x \leq wx2) & \text{&&} \\
(y \geq wy1) & \text{&&} \\
(y \leq wy2) & 
\end{cases}
\]

Line Clipping

- Find the part of a line inside the clip window

Cohen Sutherland Line Clipping

- Use simple tests to classify easy cases first

- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)
Cohen Sutherland Line Clipping

- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)

Bit 1
Bit 2
Bit 3
Bit 4

P1
P2
P3
P4
P5
P6
P7
P8
P9
P10

Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen-Sutherland Line Clipping

- Compute intersections with window boundary for lines that can't be classified quickly

Bit 1

Bit 2

Bit 3

Bit 4
Cohen-Sutherland Line Clipping

• Compute intersections with window boundary for lines that can’t be classified quickly

Bit 1  1010  0110  0101
      P_5  P_9  P_{10}

Bit 2  0010  0100  0001
      P_8  P_6  P_{10}

Bit 3  0000  0001  0000
      P_5  P_6  P_{10}

Bit 4  1011  0011  0000
      P_{10} P_6  P_8

...
Clipping

- Avoid drawing parts of primitives outside window
  - Points
  - Lines
  - Polygons
  - Circles
  - etc.

Polygon Clipping

- Find the part of a polygon inside the clip window?

Before Clipping

After Clipping

Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time
**Sutherland Hodgeman Clipping**

- Clip to each window boundary one at a time

**Clipping to a Boundary**

- Do inside test for each point in sequence, insert new points when cross window boundary, remove points outside window boundary

**Window Boundary**

- Points: P1, P2, P3, P4, P5

**Outside**

**Inside**
Clipping to a Boundary

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary

2D Rendering Pipeline

- 3D Primitives
- 2D Primitives
- Clipping
- Viewport Transformation
- Scan Conversion
- Image

Clip portions of geometric primitives residing outside the window
Transform the clipped primitives from screen to image coordinates
Fill pixels representing primitives in screen coordinates
Viewport Transformation

• Transform 2D geometric primitives from screen coordinate system (normalized device coordinates) to image coordinate system (pixels)

\[
\begin{align*}
v_x &= v_x^1 + \frac{(w_x - w_x^1)}{(w_x^2 - w_x^1)} \cdot (v_x^2 - v_x^1) \\
v_y &= v_y^1 + \frac{(w_y - w_y^1)}{(w_y^2 - w_y^1)} \cdot (v_y^2 - v_y^1)
\end{align*}
\]

Window-to-viewport mapping

Summary of Transformations

3D Primitives  \rightarrow 3D Modeling Coordinates
Modeling Transformation
\rightarrow 3D World Coordinates
Viewing Transformation
\rightarrow 3D Camera Coordinates
Projection Transformation
\rightarrow 2D Screen Coordinates
Viewport Transformation
\rightarrow 2D Image Coordinates
\rightarrow p'(x', y')

Modeling transformation

Viewing transformations

Viewport transformation

Summary

Next Time

Scan Conversion!