3D Polygon Rendering Pipeline

Adam Finkelstein
Princeton University
COS 426, Spring 2005

3D Polygon Rendering

• Many applications use rendering of 3D polygons with direct illumination

3D Polygon Rendering

• Many applications use rendering of 3D polygons with direct illumination

3D Polygon Rendering

• Many applications use rendering of 3D polygons with direct illumination

Ray Casting Revisited

• For each sample ...
  o Construct ray from eye position through view plane
  o Find first surface intersected by ray through pixel
  o Compute color of sample based on surface radiance

3D Polygon Rendering

• What steps are necessary to utilize spatial coherence while drawing these polygons into a 2D image?
This is a pipelined sequence of operations to draw a 3D primitive into a 2D image.

OpenGL executes steps of 3D rendering pipeline for each polygon.

Transform into 3D world coordinate system
Illuminate according to lighting and reflectance
Transform into 3D camera coordinate system
Illuminate according to lighting and reflectance
Transform into 2D camera coordinate system
**3D Rendering Pipeline (for direct illumination)**

Transform into 3D world coordinate system
- Modeling Transformation
- Lighting
- Viewing Transformation
- Projection Transformation
- Clipping
- Viewport Transformation
- Scan Conversion
- Image

Illuminate according to lighting and reflectance
- Modeling Transformation
- Lighting
- Viewing Transformation
- Projection Transformation
- Clipping
- Viewport Transformation
- Scan Conversion
- Image

Transform into 3D camera coordinate system
- Modeling Transformation
- Lighting
- Viewing Transformation
- Projection Transformation
- Clipping
- Viewport Transformation
- Scan Conversion
- Image

Transform into 2D camera coordinate system
- Modeling Transformation
- Lighting
- Viewing Transformation
- Projection Transformation
- Clipping
- Viewport Transformation
- Scan Conversion
- Image

Clip primitives outside camera’s view
- Modeling Transformation
- Lighting
- Viewing Transformation
- Projection Transformation
- Clipping
- Viewport Transformation
- Scan Conversion
- Image

Transform into 2D screen coordinate system
- Modeling Transformation
- Lighting
- Viewing Transformation
- Projection Transformation
- Clipping
- Viewport Transformation
- Scan Conversion
- Image

Transform into image coordinate system
- Modeling Transformation
- Lighting
- Viewing Transformation
- Projection Transformation
- Clipping
- Viewport Transformation
- Scan Conversion
- Image

**Transformations**

\[ p(x, y, z) \]

Transformations map points from one coordinate system to another

\[ p'(x', y') \]

**Viewing Transformations**

\[ p(x, y, z) \]

**Viewing Transformations**
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)
Camera right vector maps to X axis

Finding the viewing transformation

• We have the camera (in world coordinates)
• We want $T$ taking objects from world to camera

$\mathbf{p}^c = T \mathbf{p}^w$

• Trick: find $T^{-1}$ taking objects in camera to world

$\mathbf{p}^w = T^{-1} \mathbf{p}^c$

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!

Viewing Transformations

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

- In computer graphics:
  - Map 3D camera coordinates to 2D screen coordinates
Taxonomy of Projections

- 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

- Parallel Projection View Volume
**Parallel Projection Matrix**

- General parallel projection transformation:

\[
\begin{bmatrix}
    x_s \\
    y_s \\
    z_s \\
    w_s
\end{bmatrix} = \begin{bmatrix}
    1 & 0 & L \cos \phi & 0 \\
    0 & 1 & L \sin \phi & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 1 & 1
\end{bmatrix} \begin{bmatrix}
    x \\
    y \\
    z \\
    w
\end{bmatrix}
\]

**Taxonomy of Projections**

- Planar geometric projections
  - Parallel
  - Orthographic
  - Oblique
- Cabinet
- Axonometric
- Isometric
- Isometric
  - One-point perspective
  - Two-point perspective
  - Three-point perspective

**Perspective Projection**

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

**Perspective Projection View Volume**

- 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**

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

\[ (x, y, z) \rightarrow \frac{xD}{z}, \frac{yD}{z} \]

**Perspective Projection Matrix**

- 4x4 matrix representation?

\[
\begin{bmatrix}
    x' \\
y' \\
z' \\
w'
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 1/D & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
y \\
z \\
w
\end{bmatrix}
\]

**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 realistic looking
### Classical Projections

Angel Figure 5.3

### 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)

- **3D Primitives**
- **Modeling Transformation**
- **Lighting**
- **Viewing Transformation**
- **Projection Transformation**
- **Clipping**
- **Viewport Transformation**
- **Scan Conversion**
- **Image**

### 2D Rendering Pipeline

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

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

Screen Coordinates

Point Clipping

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

\[
\text{inside} = \begin{cases} 
(x \geq wx1) \land (x \leq wx2) \land (y \geq wy1) \land (y \leq wy2) & \text{if point is inside} \\
\text{false} & \text{if point is outside}
\end{cases}
\]

Line Clipping

- Find the part of a line inside the clip window

Before Clipping

After Clipping

Viewing Window
Cohen Sutherland Line Clipping

- Use simple tests to classify easy cases first

Cohen Sutherland Line Clipping

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

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
Cohen-Sutherland Line Clipping

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

0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011

Bit 1 Bit 2 Bit 3 Bit 4

- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'

Cohen-Sutherland Line Clipping

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

0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011

Bit 1 Bit 2 Bit 3 Bit 4

- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'

Cohen-Sutherland Line Clipping

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

0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011

Bit 1 Bit 2 Bit 3 Bit 4

- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'

Cohen-Sutherland Line Clipping

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

0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011

Bit 1 Bit 2 Bit 3 Bit 4

- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'

Cohen-Sutherland Line Clipping

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

0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011

Bit 1 Bit 2 Bit 3 Bit 4

- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'
- P
- P'
Cohen-Sutherland Line Clipping

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

<table>
<thead>
<tr>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1011</td>
<td>0010</td>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>1001</td>
<td>0001</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>1000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

Points

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

Polygon Clipping

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

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
- P2, P1, P3, P4, P5
- Inside: P1
- Outside: P2, P3, P4, P5
Clipping to a Boundary

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

Outside

Inside

Window
Boundary

P1
P2
P3
P4
P5

Clipping to a Boundary

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

Outside

Inside

Window
Boundary

P1
P2
P3
P4
P5

Clipping to a Boundary

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

Outside

Inside

Window
Boundary

P1
P2
P3
P4
P5

Clipping to a Boundary

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

Outside

Inside

Window
Boundary

P1
P2
P3
P4
P5

Clipping to a Boundary

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

Outside

Inside

Window
Boundary

P1
P2
P3
P4
P5

Clipping to a Boundary

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

Outside

Inside

Window
Boundary

P1
P2
P3
P4
P5

Clipping to a Boundary

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

Outside

Inside

Window
Boundary

P1
P2
P3
P4
P5

Clipping to a Boundary

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

Outside

Inside

Window
Boundary

P1
P2
P3
P4
P5

Clipping to a Boundary

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

Outside

Inside

Window
Boundary

P1
P2
P3
P4
P5
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
  - Clip portions of geometric primitives residing outside the window.
- Viewport Transformation
  - Transform the clipped primitives from screen to image coordinates.
- Scan Conversion
  - 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).

Summary of Transformations

- Modeling transformation
  - $p(x,y,z)$ → 3D Object Coordinates
  - Modeling Transformation
  - 3D World Coordinates
  - Viewing Transformation
  - 3D Camera Coordinates
  - Projection Transformation
    - 2D Screen Coordinates
    - Viewport Transformation
      - 2D Image Coordinates
      - $p'(x',y')$
  - Viewing transformations
  - Viewing Window
  - Clipping
    - 2D Screen Coordinates
    - Viewport Transformation
      - 2D Image Coordinates
      - Scan Conversion
        - Image

Summary

- Viewing Window
Next Time

3D Primitives
  3D Modeling Coordinates
    Modeling Transformation
      3D World Coordinates
        Lighting
          3D World Coordinates
            Viewing Transformation
              3D Camera Coordinates
                Projection Transformation
                  2D Screen Coordinates
                    Clipping
                      2D Screen Coordinates
                        Viewport Transformation
                          2D Image Coordinates
                            Scan Conversion
                              2D Image Coordinates
                                Image

Scan Conversion!

P
P1
P2
P3