3D Polygon Rendering Pipeline

Thomas Funkhouser
Princeton University
COS 426, Spring 2004

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

• What steps are necessary to utilize spatial coherence while drawing these polygons into a 2D image?

More efficient algorithms utilize spatial coherence!

Quake II
(Id Software)
3D Rendering Pipeline (for direct illumination)

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

Transform into 3D world coordinate system

Illuminate according to lighting and reflectance

Transform into 3D camera coordinate system

Transform into 3D world 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
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, ...)

Viewing Transformations

Transformations map points from one coordinate system to another

Transformations

3D Object Coordinates
3D World Coordinates
3D Camera Coordinates
Viewport Coordinates
2D Image Coordinates

Transformations

3D Object Coordinates
3D World Coordinates
3D Camera Coordinates
Viewport Coordinates
2D Image Coordinates

Camera Coordinates

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

Finding the viewing transformation

- We have the camera (in world coordinates)
- We want \( T \) taking objects from world to camera
  \[ p' = T p \]
- Trick: find \( T^{-1} \) taking objects in camera to world
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
  \end{bmatrix} = \begin{bmatrix}
  a & b & c & d \\
  e & f & g & h \\
  i & j & k & l \\
  m & n & o & p
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z \\
  w
  \end{bmatrix}
  \]

Viewing Transformations

- General definition:
  - Transform points in n-space to m-space (n \( \neq \) m)
- In computer graphics:
  - Map 3D camera coordinates to 2D screen coordinates

Projection
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
  - Cavalier (DOP $\alpha = 45^\circ$)
  - Cabinet (DOP $\alpha = 63.4^\circ$)

Parallel Projection View Volume
Parallel Projection Matrix

- General parallel projection transformation:

\[
\begin{bmatrix}
 x_1 \\
 x_2 \\
 x_3 \\
 w_4
\end{bmatrix} =
\begin{bmatrix}
 1 & 0 & L_1 \cos \phi & 0 \\
 0 & 1 & L_1 \sin \phi & 0 \\
 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
 x \\
 y \\
 z \\
 w
\end{bmatrix}
\]

Taxonomy of Projections

- Planar geometric projections:
  - Parallel
  - Orthographic
  - Oblique
  - One-point
  - Two-point
  - Three-point

Perspective Projection

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

Perspective Projection

- How many vanishing points?

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

Perspective Projection View Volumetric

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

Perspective Projection

- 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

![Diagram of Perspective Projection]

**Perspective Projection Matrix**

- 4x4 matrix representation?
  \[
  x' = x_d / z_v \\
  y' = y_d / z_v \\
  z' = D \\
  w' = 1
  \]

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix} = \begin{bmatrix}
  ? & ? & ? & ?
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]

**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

**Taxonomy of Projections**

- Planar geometric projections
  - Orthographic
  - Axonometric
    - Isometric
    - Cabinet
    - Oblique
  - Isometric
    - Other
  - Parallel
    - Top (plan)
    - Front elevation
    - Side elevation
  - Perspective
    - One-point
    - Two-point
    - Three-point
  - FVFHP Figure 6.10
### Classical Projections

**Front elevation**  
**Elevated oblique**  
**Isometric**  
**One-point perspective**  
**Three-point perspective**  

*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)

1. **3D Primitives**  
2. **3D Modeling Coordinates**  
3. **Transformation**  
4. **3D World Coordinates**  
5. **Lighting**  
6. **3D Camera Coordinates**  
7. **Transformation**  
8. **3D Screen Coordinates**  
9. **Clipping**
10. **2D Screen Coordinates**  
11. **Viewport Transformation**
12. **Clip portions of geometric primitives residing outside the window**
13. **Scan Conversion**
14. **2D Image Coordinates**
15. **Image**

---

### 2D Rendering Pipeline

1. **3D Primitives**  
2. **2D Primitives**  
3. **Clipping**  
4. **Viewport Transformation**  
5. **Scan Conversion**  
6. **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*
Clipping

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

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

Line Clipping

• Find the part of a line inside the clip window

Before Clipping

After Clipping
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

• 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)

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

\[
\begin{array}{c|cc|c}
\text{Bit 1} & \text{Bit 2} & \text{Bit 3} \\
1010 & 0010 & 0110 \\
1010 & 0010 & 0110 \\
1010 & 0010 & 0110 \\
1010 & 0010 & 0110 \\
\end{array}
\]

\[
\begin{array}{c|cc|c}
P_5 & P_3 & 0000 \\
P_6 & P_4 & 0100 \\
P_7 & P_5 & 0100 \\
P_8 & P_6 & 0100 \\
\end{array}
\]
Cohen-Sutherland Line Clipping

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

![Diagram of Cohen-Sutherland Line Clipping](image-url)
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

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?

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
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

P1
P2
P3
P4
P5

Inside
Outside

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

P1
P2
P3
P4
P5

Inside
Outside

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

P1
P2
P3
P4
P5

Inside
Outside

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

P1
P2
P3
P4
P5

Inside
Outside

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

P1
P2
P3
P4
P5

Inside
Outside

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

P1
P2
P3
P4
P5

Inside
Outside
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
    - Clip portions of geometric primitives residing outside the window
    - Viewport Transformation
      - 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)

Summary of Transformations

Modeling transformation
Viewing transformations
Viewport transformation

Summary

Modeling transformation
Viewing transformation
Projection transformation
Viewport transformation

Window
Viewport
Screen
Image

\[ \begin{align*}
  vx &= vx1 + (wx - wx1) \times (vx2 - vx1) / (wx2 - wx1) \\
  vy &= vy1 + (wy - wy1) \times (vy2 - vy1) / (wy2 - wy1)
\end{align*} \]
Next Time

3D Primitives
- 3D Modeling Coordinates
- 3D World Coordinates
- 3D Camera Coordinates
- 3D Screen Coordinates
- 2D Image Coordinates
- 2D Screen Coordinates

Scan Conversion!

Scan Conversion!