Rendering Pipeline
3D Polygon Rendering

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

- What steps are necessary to utilize spatial coherence while drawing these polygons into a 2D image?
**3D Rendering Pipeline (for direct illumination)**

- **Modeling Transformation**
- **Lighting**
- **Viewing Transformation**
- **Projection Transformation**
- **Clipping**
- **Viewport Transformation**
- **Scan Conversion**

**Transform** into 3D world coordinate system

Illuminate according to lighting and reflection

**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

$p(x, y, z)$

- 3D Object Coordinates
  
- 3D World Coordinates
  
- Viewing Transformation
  
- 3D Camera Coordinates
  
- Projection Transformation
  
- 2D Screen Coordinates
  
- Viewport Transformation
  
- 2D Image Coordinates
  
$p'(x', y')$

Transformations map points from one coordinate system to another.
Viewing Transformations

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

- 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 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.

![Diagram showing canonical camera coordinates with axes and vector directions]
Finding the viewing transformation

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

$$p^c = Tp^w$$

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

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

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

- **Trick: map from camera coordinate 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

\[
p^w_w = T^{-1} p^c_c
\]

- This matrix is $T^{-1}$ so we invert it to get $T$ (easy)
Viewing Transformations

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

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Viewing Transformation

3D Camera Coordinates

Projection Transformation

2D Screen Coordinates

Viewport Transformation

2D Image Coordinates

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

- **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

Planar geometric projections

Parallel

Orthographic
- Top (plan)
- Front elevation
- Side elevation
- Axonometric
- Isometric

Oblique
- Cabinet
- Cavalier

Perspective

One-point
- Two-point
- Three-point

Other
Parallel Projection

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

- DOP perpendicular to view plane
**Oblique Projections**

- **DOP not perpendicular to view plane**
  1) Cavalier projection
     - when angle between projector and view plane is 45 degree
     - foreshortening ratio for all three principal axis are equal.
  2) Cabinet projection
     - the foreshortening ratio for edges that perpendicular to the plane of projection is one-half (angle 63.43 degree)

![Diagram of oblique projections](image)
Oblique Projections

- DOP not perpendicular to view plane

- $\phi$ describes the angle of the projection of the view plane’s normal
- $L$ represents the scale factor applied to the view plane’s normal

Cavalier
(DOP $\alpha = 45^\circ$)

Cabinet
(DOP $\alpha = 63.4^\circ$)
Parallel Projection View Volume
Parallel Projection Matrix

- General parallel projection transformation

\[
\begin{bmatrix}
    x_s \\
    y_s \\
    z_s \\
    w_s
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & L_1 \cos \phi & 0 \\
    0 & 1 & L_1 \sin \phi & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x_c \\
    y_c \\
    z_c \\
    1
\end{bmatrix}
\]
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 Volume
Perspective Projection

- Compute 2D coordinates from 3D coordinates with similar triangles
 Perspective Projection Matrix

4 x 4 matrix representation?

\[
x_s = x_c \frac{D}{z_c}
\]
\[
y_s = y_c \frac{D}{z_c}
\]
\[
z_s = D
\]
\[
w_s = 1
\]
4 x 4 matrix representation?

\[
\begin{align*}
x_s &= x_c \frac{D}{z_c} \\
y_s &= y_c \frac{D}{z_c} \\
z_s &= D \\
w_s &= 1
\end{align*}
\]

\[
\begin{bmatrix}
x_s \\
y_s \\
z_s \\
w_s
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1/D & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x_c \\
y_c \\
z_c \\
1
\end{bmatrix}
\]
**Perspective vs. Parallel**

- **Parallel Projection**
  - Good for exact measurements
  - Parallel lines remain parallel
  - Angles are not (in general) preserved
  - Less realistic looking

- **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
Classical Projections

- Front elevation
- Elevation oblique
- Plan oblique
- Isometric
- One-point perspective
- Three-point perspective
Taxonomy of Projections

Planar geometric projections

Parallel
- Orthographic
  - Top (plan)
  - Front elevation
  - Side elevation
- Axonometric
  - Isometric

Oblique
- Cabinet
- Cavalier

Perspective
- One-point
  - Two-point
  - Three-point
- Other
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)
2D Rendering Pipeline

3D Primitives

→

2D Primitives

Clip portions of geometric primitives residing outside the window

Clipping

→

Viewport Transformation

Transform the clipped primitives From screen to image coordinates

→

Scan Conversion

Fill pixels representing primitives In screen coordinates

→

Image
Clipping

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

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

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

\[
\text{inside} =
\begin{align*}
(x & \geq wx1) \& \& \\
(x & \leq wx2) \& \& \\
y & \geq wy1 \& \& \\
y & \leq wy2) \\
\end{align*}
\]
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 AND of bit codes representing regions of two end points (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
Polygon Clipping

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

After Clipping
Sutherland Hodgeman Clipping

- Clip each window boundary one at a time
1. Do inside test for each point in sequence,
2. Insert new points when cross window boundary,
3. 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

Image
Viewport Transformation

- Transformation 2D geometric primitives from screen coordinate system (normalized device coordinates) to image coordinate system (pixels)
Viewport Transformation

- Window-to-Viewport mapping

\[
\begin{align*}
vx &= vx_1 + (wx - wx_1) \times (vx_2 - vx_1) / (wx_2 - wx_1); \\
vy &= vy_1 + (wy - wy_1) \times (vy_2 - vy_1) / (wy_2 - wy_1);
\end{align*}
\]
Summary of Transformations

1. 3D Object Coordinates
   - Modeling Transformation
     - 3D World Coordinates
       - Viewing Transformation
         - 3D Camera Coordinates
           - Projection Transformation
             - 2D Screen Coordinates
               - Viewport Transformation
                 - 2D Image Coordinates
                   - \( p'(x', y') \)

- Modeling transformation
- Viewing transformations
- Viewport transformation
Summary

3D Primitives
  \rightarrow \text{Modeling Transformation}
  \rightarrow \text{Lighting}
  \rightarrow \text{Viewing Transformation}
  \rightarrow \text{Projection Transformation}
  \rightarrow \text{Clipping}
  \rightarrow \text{Viewport Transformation}
  \rightarrow \text{Scan Conversion}
  \rightarrow \text{Image}

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

Viewing Window
Next Time

3D Primitives

Modeling Transformation

3D Modeling Coordinates

Lighting

3D World Coordinates

Viewing Transformation

3D World Coordinates

Projection Transformation

3D Camera Coordinates

Clipping

2D Screen Coordinates

Viewport Transformation

2D Screen Coordinates

Scan Conversion

2D Image Coordinates

Image

Scan Conversion!