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

• 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

This is a pipelined sequence of operations to draw a 3D primitive into a 2D image.
GPU Architecture

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

glBegin(GL_POLYGON);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glVertex3f(1.0, 1.0, 0.0);
glVertex3f(0.0, 1.0, 1.0);
glEnd();
3D Rendering Pipeline (for direct illumination)

Transform into 3D world 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

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

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
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

- p(x,y,z)
- 3D Object Coordinates
- Modeling Transformation
- 3D World Coordinates
- Viewing Transformation
- 3D Camera Coordinates
- Projection Transformation
- 2D Screen Coordinates
- Viewing 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.

- Camera up vector maps to Y axis
- Camera back vector maps to Z axis
- 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
  \[ p' = Tp \]
- Trick: find \( T^{-1} \) taking objects in camera to world
  \[ p = T^{-1}p' \]

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
  \[ \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ w' \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \]

- 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

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
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 transformation:

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

Taxonomy of Projections
Perspective Projection

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

**Center of Projection**

![Image](Angel_Figure_5.9)

Perspective Projection View Volume

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

**View Plane**

![Image](H&B_Figure_12.30)

Perspective Projection

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

![Image](Angel_Figure_5.10)

Perspective Projection Matrix

- 4x4 matrix representation?

\[
\begin{bmatrix}
    x_s \\
    y_s \\
    z_s \\
    w_s
\end{bmatrix} = \begin{bmatrix}
    ? & ? & ? & x_s \\
    ? & ? & ? & y_s \\
    ? & ? & ? & z_s \\
    ? & ? & ? & 1
\end{bmatrix}
\]
**Perspective Projection Matrix**

- 4x4 matrix representation?

\[
\begin{pmatrix}
  x' \\
y' \\
z' \\
w'
\end{pmatrix} = \frac{\begin{pmatrix}
x \\
y \\
z \\
w
\end{pmatrix}}{w} = \frac{\begin{pmatrix}
x \\
y \\
z \\
w
\end{pmatrix}}{z/D}
\]

\[
\begin{pmatrix}
x' \\
y' \\
z' \\
w'
\end{pmatrix} = \begin{pmatrix}
x \\
y \\
z \\
w
\end{pmatrix}
\]

**Taxonomy of Projections**

- Parallel geometry:

  - Planar geometric projections
    - Parallel
    - Orthographic
    - Top (plan)
    - Front elevation
    - Axonometric
    - Isometric
  - Perspective
    - One-point
    - Two-point
    - Three-point
  - Other

**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**

- Front elevation
- Elevations oblique
- Plan oblique
- Isometric
- One-point perspective
- Three-point perspective

**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)

Clipping

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

2D Rendering Pipeline

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

3D Primitives

Modeling

World Coordinates

Lighting

Camera Coordinates

Projection

Screen Coordinates

Viewport

Screen Coordinates

Clipping

Screen Coordinates

Image

Clipping

Screen Coordinates

Image
Clipping

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

Point Clipping

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

Line Clipping

- Find the part of a line inside the clip 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)

\[
\begin{array}{c|c|c|c}
\text{Bit 4} & \text{Bit 3} & \text{Bit 2} & \text{Bit 1} \\
1011 & 0001 & 0010 & 1010 \\
1000 & 0000 & 0011 & 1010 \\
1010 & 0000 & 0010 & 1010 \\
\end{array}
\]

Cohen-Sutherland Line Clipping

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

\[
\begin{array}{c|c|c|c}
\text{Bit 4} & \text{Bit 3} & \text{Bit 2} & \text{Bit 1} \\
1011 & 0001 & 0010 & 1010 \\
1000 & 0000 & 0011 & 1010 \\
1010 & 0000 & 0010 & 1010 \\
\end{array}
\]

Cohen-Sutherland Line Clipping

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

\[
\begin{array}{c|c|c|c}
\text{Bit 4} & \text{Bit 3} & \text{Bit 2} & \text{Bit 1} \\
1011 & 0001 & 0010 & 1010 \\
1000 & 0000 & 0011 & 1010 \\
1010 & 0000 & 0010 & 1010 \\
\end{array}
\]

Cohen-Sutherland Line Clipping

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

\[
\begin{array}{c|c|c|c}
\text{Bit 4} & \text{Bit 3} & \text{Bit 2} & \text{Bit 1} \\
1011 & 0001 & 0010 & 1010 \\
1000 & 0000 & 0011 & 1010 \\
1010 & 0000 & 0010 & 1010 \\
\end{array}
\]

Cohen-Sutherland Line Clipping

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

\[
\begin{array}{c|c|c|c}
\text{Bit 4} & \text{Bit 3} & \text{Bit 2} & \text{Bit 1} \\
1011 & 0001 & 0010 & 1010 \\
1000 & 0000 & 0011 & 1010 \\
1010 & 0000 & 0010 & 1010 \\
\end{array}
\]

Cohen-Sutherland Line Clipping

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

\[
\begin{array}{c|c|c|c}
\text{Bit 4} & \text{Bit 3} & \text{Bit 2} & \text{Bit 1} \\
1011 & 0001 & 0010 & 1010 \\
1000 & 0000 & 0011 & 1010 \\
1010 & 0000 & 0010 & 1010 \\
\end{array}
\]

Cohen-Sutherland Line Clipping

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

\[
\begin{array}{c|c|c|c}
\text{Bit 4} & \text{Bit 3} & \text{Bit 2} & \text{Bit 1} \\
1011 & 0001 & 0010 & 1010 \\
1000 & 0000 & 0011 & 1010 \\
1010 & 0000 & 0010 & 1010 \\
\end{array}
\]

Cohen-Sutherland Line Clipping

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

\[
\begin{array}{c|c|c|c}
\text{Bit 4} & \text{Bit 3} & \text{Bit 2} & \text{Bit 1} \\
1011 & 0001 & 0010 & 1010 \\
1000 & 0000 & 0011 & 1010 \\
1010 & 0000 & 0010 & 1010 \\
\end{array}
\]
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

---

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

1001 | 0001 | 0101
-----|-------|------
  P_3 | P'_{10} | Bit 4

1000 | 0000 | 0100
-----|-------|------
  P_4 | P_{10} | Bit 3

1010 | 0010 | 0110
-----|-------|------
  P_5 | P'_{10} | Bit 3

Bit 1 | Bit 2 | Bit 3 | Bit 4
-----|-------|------|------
Clipping

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

2D Screen Coordinates

Polygon Clipping

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

Before Clipping

Polygon Clipping

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

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

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

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

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)

Summary of Transformations

- Modeling transformation
- Viewing transformations
- Viewport transformation

Next Time

Window-to-viewpoint mapping

\[
\begin{align*}
wx &= w_1 + (w_2 - w_1) \times (wx - w_1) / (w_2 - w_1); \\
wy &= w_1 + (w_y - w_1) \times (wy - w_1) / (w_2 - w_1);
\end{align*}
\]