3D Polygon Rendering Pipeline

Michael Kazhdan
(601.457/657)

HB Ch. 12
FvDFH Ch. 6, 18.3
3D Polygon Rendering

- Many applications require interactive rendering of 3D polygons with direct illumination

God of War
(Santa Monica Studio, 2018)
Ray Casting

• For each sample:
  ○ Construct ray from the camera into the scene
  ○ Find first surface intersected by ray through pixel
  ○ Compute color of sample based on surface radiance
  ↓
  ○ Send 2D pixels into the scene and get color
3D Polygon Rendering

• For each primitive:
  ◦ Send 3D points to the camera and set the pixel color
3D Rendering Pipeline (for direct illumination)

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

3D Model

2D Viewport
3D Rendering Pipeline (for direct illumination)

1. Modeling Transformation
2. Viewing Transformation
3. Lighting
4. Projection Transformation
5. Clipping
6. Viewport Transformation
7. Scan Conversion
8. Image

Transform into 3D world coordinate system
3D Rendering Pipeline (for direct illumination)

3D Primitives

Transform into 3D world coordinate system

Modeling Transformation

3D Modeling Coordinates

Viewing Transformation

Transform into 3D camera coordinate system

3D World Coordinates

Lighting

Projection Transformation

3D Camera Coordinates

Clipping

Viewport Transformation

Scan Conversion

Image
3D Rendering Pipeline (for direct illumination)

- **3D Primitives**
- **Modeling Transformation**
  - Transform into 3D modeling coordinates
- **Viewing Transformation**
  - Transform into 3D world coordinate system
  - Transform into 3D camera coordinate system
- **Lighting**
  - Illuminate *vertices* using lighting and reflectance
- **Projection Transformation**
  - 3D Camera Coordinates
- **Clipping**
- **Viewport Transformation**
- **Scan Conversion**
- **Image**
3D Rendering Pipeline (for direct illumination)

- 3D Primitives
  - Modeling Transformation
    - 3D Modeling Coordinates
    - Transform into 3D world coordinate system
    - 3D World Coordinates
    - Transform into 3D camera coordinate system
    - 3D Camera Coordinates
  - Viewing Transformation
    - 3D Camera Coordinates
    - Illuminate vertices using lighting and reflectance
    - 3D Camera Coordinates
  - Lighting
    - 3D Camera Coordinates
    - Transform into 2D window coordinate system
  - Projection Transformation
  - Clipping
  - Viewport Transformation
  - Scan Conversion
  - Image
3D Rendering Pipeline (for direct illumination)

- **3D Primitives**
  - 3D Modeling Coordinates
    - Transform into 3D world coordinate system
  - 3D World Coordinates
    - Transform into 3D camera coordinate system
  - 3D Camera Coordinates
    - Illuminate vertices using lighting and reflectance
  - 3D Camera Coordinates
    - Transform into 2D window coordinate system
  - 2D Window Coordinates
    - Clip (parts of) primitives outside camera’s view
  - 2D Window Coordinates
    - Image
3D Rendering Pipeline (for direct illumination)

3D Primitives

- **Modeling Transformation**
  - Transform into 3D world coordinate system
  - 3D Modeling Coordinates

- **Viewing Transformation**
  - Transform into 3D camera coordinate system
  - 3D World Coordinates

- **Lighting**
  - Illuminate vertices using lighting and reflectance
  - 3D Camera Coordinates

- **Projection Transformation**
  - Transform into 2D window coordinate system
  - 3D Camera Coordinates

- **Clipping**
  - Clip (parts of) primitives outside camera’s view
  - 2D Window Coordinates

- **Viewport Transformation**
  - Transform into 2D viewport coordinate system
  - 2D Window Coordinates

- **Scan Conversion**
  - 2D Viewport Coordinates

- **Image**
3D Rendering Pipeline (for direct illumination)

3D Primitives

- Modeling Transformation
  - Transform into 3D world coordinate system

- Viewing Transformation
  - Transform into 3D camera coordinate system

- Lighting
  - Illuminate vertices using lighting and reflectance

- Projection Transformation
  - Transform into 2D window coordinate system

- Clipping
  - Clip (parts of) primitives outside camera’s view

- Viewport Transformation
  - Transform into 2D viewport coordinate system

- Scan Conversion
  - Draw pixels (includes texturing, hidden surface, etc.)
Transformations

3D Primitives

Modeling Transformation

Transform into 3D modeling coordinates

Viewing Transformation

Transform into 3D world coordinate system

Lighting

Illuminate vertices using lighting and reflectance

Projection Transformation

Transform into 3D camera coordinate system

Clipping

Clip (parts of) primitives outside camera’s view

Viewport Transformation

Transform into 2D window coordinate system

Scan Conversion

Transform into 2D viewport coordinate system

Draw pixels (includes texturing, hidden surface, etc.)
Recall: Homogeneous Coordinates

• Add a 4th coordinate to every 3D point
  ◦ \((x, y, z, w)\) represents a point at location \(\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)\)
  ◦ \((x, y, z, 0)\) represents the (unsigned) direction \(\frac{\pm(x, y, z)}{\sqrt{x^2 + y^2 + z^2}}\)
  ◦ \((0, 0, 0,0)\) is not allowed
Recall: 3D Transformations

- Using homogenous coordinates, we have two types of transformations:
  - Affine
    \[
    \begin{bmatrix}
      x' \\
      y' \\
      z' \\
      1
    \end{bmatrix} =
    \begin{bmatrix}
      a & b & c & d \\
      e & f & g & h \\
      i & j & k & l \\
      0 & 0 & 0 & 1
    \end{bmatrix}
    \begin{bmatrix}
      x \\
      y \\
      z \\
      1
    \end{bmatrix}
    \]
  - Projective
    \[
    \begin{bmatrix}
      x' \\
      y' \\
      z' \\
      w'
    \end{bmatrix} =
    \begin{bmatrix}
      a & b & c & d \\
      e & f & g & h \\
      i & j & k & l \\
      m & n & o & p
    \end{bmatrix}
    \begin{bmatrix}
      x \\
      y \\
      z \\
      w
    \end{bmatrix}
    \]
Transformations

\[(x, y, z)\]

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Viewing Transformation

3D Camera Coordinates

Projection Transformation

2D Window Coordinates

Window-to-Viewport Transformation

2D Viewport Coordinates

\[(x', y')\]

Transformations map points from one coordinate system to another.
Transformations

\[(x, y, z)\]

- 3D Object Coordinates
- 3D World Coordinates
- 3D Camera Coordinates
- 2D Window Coordinates
- 2D Viewport Coordinates

Modelview Transformations
Transformations

\[(x, y, z)\]

- **Modeling Transformation**
  - 3D Object Coordinates

- **Viewing Transformation**
  - 3D World Coordinates
  - 3D Camera Coordinates

- **Projection Transformation**
  - 2D Window Coordinates

- **Window-to-Viewport Transformation**
  - 2D Viewport Coordinates

\[(x', y')\]
Viewing Transformation

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

• The transformation, $T_{W\rightarrow C}$, taking us from world coordinates to camera coordinates should map:
  ○ The right vector to the $x$-axis:
    \[(R_x, R_y, R_z, 0) \rightarrow (1,0,0,0)\]
  ○ The up vector to the $y$-axis:
    \[(U_x, U_y, U_z, 0) \rightarrow (0,1,0,0)\]
  ○ The back vector to the $z$-axis:
    \[(B_x, B_y, B_z, 0) \rightarrow (0,0,1,0)\]
  ○ The eye position to the origin:
    \[(E_x, E_y, E_z, 1) \rightarrow (0,0,0,1)\]

How should we define this transformation/matrix?
Viewing Transformation

• Consider the inverse transformation, $T_{C \rightarrow W}$, taking us from camera coordinates to world coordinates:

\[
\begin{align*}
(R_x, R_y, R_z, 0) &\leftarrow (1,0,0,0) \\
(U_x, U_y, U_z, 0) &\leftarrow (0,1,0,0) \\
(B_x, B_y, B_z, 0) &\leftarrow (0,0,1,0) \\
(E_x, E_y, E_z, 1) &\leftarrow (0,0,0,1)
\end{align*}
\]

• This is described by the matrix:

\[
\begin{pmatrix}
x^w \\
y^w \\
z^w \\
1
\end{pmatrix}
= 
\begin{pmatrix}
R_x & U_x & B_x & E_x \\
R_y & U_y & B_y & E_y \\
R_z & U_z & B_z & E_z \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x^c \\
y^c \\
z^c \\
1
\end{pmatrix}
\]

$T_{C \rightarrow W}$
Finding the Viewing Transformation

- The camera-to-world matrix:

\[
\begin{pmatrix}
    x^w \\
y^w \\
z^w \\
1
\end{pmatrix} = \begin{pmatrix}
    R_x & U_x & B_x & E_x \\
    R_y & U_y & B_y & E_y \\
    R_z & U_z & B_z & E_z \\
    0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    x^c \\
y^c \\
z^c \\
1
\end{pmatrix}
\]

\[T_{C\rightarrow W}\]

- The world-to-camera matrix is its inverse:

\[
\begin{pmatrix}
    x^c \\
y^c \\
z^c \\
1
\end{pmatrix} = \begin{pmatrix}
    R_x & U_x & B_x & E_x \\
    R_y & U_y & B_y & E_y \\
    R_z & U_z & B_z & E_z \\
    0 & 0 & 0 & 1
\end{pmatrix}^{-1}
\begin{pmatrix}
    x^w \\
y^w \\
z^w \\
1
\end{pmatrix}
\]

\[T_{W\rightarrow C} = T_{C\rightarrow W}^{-1}\]
Transformations

\((x, y, z)\)

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Viewing Transformation

3D Camera Coordinates

Projection Transformation

2D Window Coordinates

Window-to-Viewport Transformation

2D Viewport Coordinates

\((x', y')\)
Taxonomy of Projections

Planar geometric projections

Parallel

Orthographic

Top (plan)
Front elevation
Side elevation
 Axonometric
 Isometric

Oblique
 Cabinet
 Cavalier

One-point
 Two-point
 Three-point

Perspective

Other

FvDFH Figure 6.13
Projection

- Two general classes of projections, which shoot rays from the 3D scene, through the 2D window:
  - Parallel Projection:
    » Rays converge at a point at infinity and are parallel
  - Perspective “Projection”:
    » Rays converge at a finite point, giving rise to perspective distortion
Taxonomy of Projections

Planar geometric projections

Parallel

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

Oblique
- Cabinet
- Cavalier

One-point
- Two-point
- Three-point

Perspective
- Other

FvDFH Figure 6.13
Parallel Projection

- Center of projection is at infinity
  - Direction of projection (DoP) same for all points
Parallel Projection

✓ Parallel lines remain parallel
✓ Proportions are preserved (no foreshortening)
✗ Angles are not preserved
✗ Less realistic looking
Taxonomy of Projections

Planar geometric projections

Parallel

Orthographic

Top (plan)
Front elevation
Side elevation
Axonometric

Oblique
Cabinet
Cavalier

Perspective
One-point
Two-point
Three-point

Other
Other
Orthographic Projections

- DoP perpendicular to view plane
Orthographic Projections

- DoP perpendicular to view plane

- Lines perpendicular to the view plane vanish
- Faces parallel to the view plane are un-distorted.
Orthographic Projections

• DoP perpendicular to view plane
  - Maps a point in 3D space to the \((x, y, -1)\)-plane, by projecting out the \(z\)-component:
    \[(x^c, y^c, z^c) \rightarrow (x^c, y^c, -1)\]
  - In terms of homogenous coordinates:
    \[
    \begin{bmatrix}
    x^c \\
    y^c \\
    -1 \\
    1
    \end{bmatrix} =
    \begin{bmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 0 & -1 \\
    0 & 0 & 0 & 1
    \end{bmatrix}
    \begin{bmatrix}
    x^c \\
    y^c \\
    z^c \\
    1
    \end{bmatrix}
    \]
Orthographic Projections

• DoP perpendicular to view plane
  ◦ Maps a point in 3D space to the \((x, y, -1)\)-plane, by projecting out the \(z\)-component:

\[
(x^c, y^c, z^c) \rightarrow (x^c, y^c, -1)
\]

Note:
This matrix describes an affine transformation

\[
\begin{bmatrix}
-1 & 0 & 0 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1
\end{bmatrix}
\]

\((x^c, y^c, z^c)\)
Direction of projection not perpendicular to view plane.

FvDFH Figure 6.13
Parallel Projection View Volume

Parallelepiped View Volume

Back Plane

Front Plane

window

$Z_V$

H&B Figure 12.30
Taxonomy of Projections

Planar geometric projections

Parallel

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

Oblique
- Cabinet
  - Cavalier

One-point
- Two-point
- Three-point

Perspective

Other
Perspective “Projection”

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

- Not all parallel lines remain parallel!
Perspective Projection

• What are the coordinates of the point resulting from projection of \((x^c, y^c, z^c)\) onto the camera screen plane a unit distance along the \(z\)-axis?
Perspective Projection

• For any point \((x^c, y^c, z^c)\) and any scalar \(\alpha\), the points \((x^c, y^c, z^c)\) and \((\alpha x^c, \alpha y^c, \alpha z^c)\) map to the same location.

\((0,0,0)\)
Perspective Projection

- For any point \((x^c, y^c, z^c)\) and any scalar \(\alpha\), the points \((x^c, y^c, z^c)\) and \((\alpha x^c, \alpha y^c, \alpha z^c)\) map to the same location.

- Since we want the position on the window that intersects the line from \((x^c, y^c, z^c)\) to the origin:

\[
(x^c, y^c, z^c) \rightarrow \left( \frac{x^c}{-z^c}, \frac{y^c}{-z^c}, -1 \right)
\]
Perspective Projection Matrix

\[(x^c, y^c, z^c) \rightarrow \left( \frac{x^c}{-z^c}, \frac{y^c}{-z^c}, -1 \right)\]

We can’t represent this with a 3 × 3 matrix!

With homogenous coordinates, we can write this as:

\[(x^c, y^c, z^c, 1) \rightarrow \left( \frac{x^c}{-z^c}, \frac{y^c}{-z^c}, -1, 1 \right) \equiv (x^c, y^c, z^c, -z^c)\]

In matrix form, this gives:

\[
\begin{bmatrix}
-x^c/z^c \\
-y^c/z^c \\
-1 \\
1
\end{bmatrix} \equiv \begin{bmatrix}
x^c \\
y^c \\
z^c \\
-z^c
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1 & 0
\end{bmatrix} \begin{bmatrix}
x^c \\
y^c \\
z^c \\
1
\end{bmatrix}
\]
Perspective Projection Matrix

\[(x^c, y^c, z^c) \rightarrow \left( \frac{x^c}{-z^c}, \frac{y^c}{-z^c}, -1 \right)\]

We can’t represent this with a \(3 \times 3\) matrix!

With homogenous coordinates, we can write this as:

\[(x^c, y^c, z^c, 1) \rightarrow \left( \frac{x^c}{-z^c}, \frac{y^c}{-z^c}, -1, 1 \right) \equiv (x^c, y^c, z^c, -z^c)\]

Note: This matrix describes a projective transformation

\[
\begin{bmatrix}
-x^c / z^c \\
-y^c / z^c \\
-1 \\
1
\end{bmatrix}
\equiv
\begin{bmatrix}
x^c \\
y^c \\
z^c \\
-z^c
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & -1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x^c \\
y^c \\
z^c \\
1
\end{bmatrix}
\]
Perspective Projection View Volume

H&B Figure 12.30
Taxonomy of Projections

Planar geometric projections

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

Oblique
- Cabinet
- Cavalier
- Other

Perspective
- One-point
- Two-point
- Three-point

FVFHP Figure 6.10
Classical Projections

Front elevation  Elevation oblique  Plan oblique

Isometric  One-point perspective  Three-point perspective

Angel Figure 5.3
Perspective vs. Parallel

- **Perspective projection**
  - ✓ Size varies inversely with distance - looks realistic
  - ✓ Angles are preserved on faces parallel to the view plane
  - ✗ Distance are not preserved

- **Parallel (orthographic) projection**
  - ✓ Parallel lines remain parallel
  - ✓ Angles and distance are preserved on faces parallel to the view plane
  - ✗ Less realistic looking
  - ✓ Good for exact measurements
Transformations

\((x, y, z)\)

\(\text{Modeling Transformation}\)

\(3D\ Object\ Coordinates\)

\(\text{Viewing Transformation}\)

\(3D\ World\ Coordinates\)

\(\text{Projection Transformation}\)

\(3D\ Camera\ Coordinates\)

\(\text{Window-to-Viewport Transformation}\)

\(2D\ Window\ Coordinates\)

\(\text{Window Coordinates}\)

\((w_x, w_y)\)

\(\text{Viewport Coordinates}\)

\((v_x, v_y)\)

\(V = \text{viewport transform}\)

\[
V = \begin{bmatrix}
1 & 0 & v_x^1 \\
0 & 1 & w_x^2 - w_x^1 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
v_x^2 - v_x^1 \\
w_y^2 - w_y^1 \\
0
\end{bmatrix}
\]

\[
V = \begin{bmatrix}
1 & 0 & -w_x^1 \\
0 & 1 & -w_y^1 \\
0 & 0 & 1
\end{bmatrix}
\]

Note that this may scale non-uniformly.
3D Rendering Pipeline (for direct illumination)

\((x, y, z)\)

- **Modeling Transformation**
  - 3D Object Coordinates
- **Viewing Transformation**
  - 3D World Coordinates
- **Projection Transformation**
  - 3D Camera Coordinates
- **Window-to-Viewport Transformation**
  - 2D Window Coordinates
  - 2D Viewport Coordinates

\((x', y')\)

- 3D Model
- 2D Viewport
Transformations

3D Primitives → 3D Modeling Coordinates
Modeling Transformation

3D Modeling Coordinates → 3D World Coordinates
Viewing Transformation

3D World Coordinates → 3D Camera Coordinates
Lighting

3D Camera Coordinates → 3D Camera Coordinates
Projection Transformation

3D Camera Coordinates → 2D Window Coordinates
Clipping

2D Window Coordinates → 2D Window Coordinates
Viewport Transformation

2D Window Coordinates → 2D Viewport Coordinates
Scan Conversion

2D Viewport Coordinates → 2D Viewport Coordinates
Image

\[ I = I_E + \sum_L \left( K_A \cdot I_L^A + (K_D \cdot \langle \vec{N}, \vec{L} \rangle + K_S \cdot \langle \vec{V}, \vec{R} \rangle^n) \cdot I_L \right) \]
Transformations

- Originally, vertex processing was fixed
- Now this is programmable in the vertex shader
Clipping

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

- Avoid drawing parts of primitives outside window
  - Points
  - Line Segments
  - Polygons
Point Clipping

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

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

\[
\text{inside} = (x \geq x_{\text{min}}) \land (x \leq x_{\text{max}}) \land (y \geq y_{\text{min}}) \land (y \leq y_{\text{max}});
\]
Clipping

- Avoid drawing parts of primitives outside window
  - Points
  - **Line Segments**
  - Polygons
Line Segment Clipping

• Find the part of a line inside the clip window
  ◦ Do this as **efficiently** as possible by identifying the easiest cases first
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0 b_1 b_2 b_3$ to each vertex
  - $b_0 = 1$ if the vertex is left of the window
  - $b_1 = 1$ if the vertex is right of the window
  - $b_2 = 1$ if the vertex is below the window
  - $b_3 = 1$ if the vertex is above the window

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>0001</td>
<td>0101</td>
</tr>
<tr>
<td>1000</td>
<td>0000</td>
<td>0100</td>
</tr>
<tr>
<td>1010</td>
<td>0010</td>
<td>0110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0 b_1 b_2 b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode \( b_0 b_1 b_2 b_3 \) to each vertex.
- If both points’ outcodes are 0, line segment is inside.
- If AND of outcodes is not 0, line segment is outside.
- Otherwise clip and test.

![Diagram of Cohen-Sutherland Line Clipping]

- Points: \( P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10} \)
- Outcodes: \( b_0, b_1, b_2, b_3 \)

\[ \begin{array}{cccc}
1001 & P_7 & 0001 & 0101 \\
1000 & P_3 & 0000 & 0100 \\
1010 & P_5 & 0010 & 0110 \\
\end{array} \]
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode \( b_0 b_1 b_2 b_3 \) to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test

\[
\begin{array}{c|c|c|c|c}
0000 & 0001 & 0010 & 0011 \\
0100 & 0101 & 0110 & 0111 \\
1000 & 1001 & 1010 & 1011 \\
1100 & 1101 & 1110 & 1111 \\
\end{array}
\]

- \( P_0 \) to \( P_7 \)
- \( P_8 \) to \( P_{10} \)
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode \( b_0 b_1 b_2 b_3 \) to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test

\[ b_0 \quad b_1 \quad b_2 \quad b_3 \]

\[ P_5 \quad 0010 \]
\[ P_6 \quad P_7 \quad 0001 \]
\[ P_8 \quad 0100 \]
\[ P_9 \quad 0110 \]
\[ P_{10} \]
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0 b_1 b_2 b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test

<table>
<thead>
<tr>
<th>$b_0$</th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0010</td>
<td>0010</td>
<td>0101</td>
</tr>
<tr>
<td>1001</td>
<td>0001</td>
<td>0100</td>
<td>0110</td>
</tr>
<tr>
<td>1010</td>
<td>0010</td>
<td>0100</td>
<td>0110</td>
</tr>
<tr>
<td>1010</td>
<td>0010</td>
<td>0100</td>
<td>0110</td>
</tr>
</tbody>
</table>

Points: $P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}$
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode \( b_0b_1b_2b_3 \) to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test
Cohen-Sutherland Line Clipping

• Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
• If both points’ outcodes are 0, line segment is inside
• If AND of outcodes is not 0, line segment is outside
• Otherwise clip and test

$P_8' \quad P_7' \quad P_6 \quad P_3 \quad P_4 \quad P_5'$
Cohen-Sutherland Line Clipping

- Associate a 4-bit outcode $b_0b_1b_2b_3$ to each vertex
- If both points’ outcodes are 0, line segment is inside
- If AND of outcodes is not 0, line segment is outside
- Otherwise clip and test

How many bits would you need in 3D?
Clipping

- Avoid drawing parts of primitives outside window
  - Points
  - Line Segments
  - Polygons
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
Sutherland-Hodgeman Clipping

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

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

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

• How do we clip a convex polygon with respect to a (window boundary) line?
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.

![Diagram showing Sutherland-Hodgeman Clipping process with points P1, P2, P3, P4, P5, and P']
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.
Sutherland-Hodgeman Clipping

• Do interior test for each point in sequence.
• Insert a new point when crossing the line.
• Remove points outside the line.
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.

When polygons are clipped, per-vertex properties (e.g. lighting) is interpolated to the new vertices.
Sutherland-Hodgeman Clipping

- Do interior test for each point in sequence.
- Insert a new point when crossing the line.
- Remove points outside the line.

[WARNING] If the polygon is not convex, we may end up with more than one polygon!
At this point we have the:

- Positions of the mesh vertices (including new vertices obtained through clipping)
- Color information at each vertex.
- A list of (possibly clipped) polygons describing the intersection of the projected 3D polygons with the window.