Clipping
3D Rendering Pipeline

3D Geometric Primitives

- **Modeling Transformation**: Transform into 3D world coordinate system
- **Lighting**: Illuminate according to lighting and reflectance
- **Viewing Transformation**: Transform into 3D camera coordinate system
- **Projection Transformation**: Transform into 2D camera coordinate system
- **Clipping**: Clip primitives outside camera’s view
- **Scan Conversion**: Draw pixels (includes texturing, hidden surface, etc.)
- **Image**
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
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
  - 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 the clip window?

\[
\text{inside} = (x \geq wx1) \&\& (x \leq wx2) \&\& (y \geq wy1) \&\& (y \leq wy2);
\]
Line Clipping

- Find the part of a line inside the clip window

Before Clipping
Line Clipping

- Find the part of a line inside the clip window
Cohen Sutherland Line Clipping

- Use simple tests to classify easy cases first
Key Idea: Encode Regions

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Bit 1 (left)</th>
<th>Bit 2 (right)</th>
<th>Bit 3 (down)</th>
<th>Bit 4 (up)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x &lt; x_{\text{min}}$</td>
<td>1001</td>
<td>0001</td>
<td>0101</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$x &gt; x_{\text{max}}$</td>
<td>1000</td>
<td>0000</td>
<td>0100</td>
<td>0110</td>
</tr>
<tr>
<td>3</td>
<td>$y &lt; y_{\text{min}}$</td>
<td>1010</td>
<td>0010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$y &gt; y_{\text{max}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is order important?
Endpoint Code Classification

- Assign code to end points of each line segment $c_1$, $c_2$
  - line is inside if ($c_1 \lor c_2 == 0$)
  - line is outside if ($c_1 \land c_2 \neq 0$)
- Remove all such lines
- Compute intersections with window boundary for all lines that can’t be classified quickly
  - Generate new points and repeat
Cohen Sutherland Line Clipping

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

Bit 1

Bit 2

Bit 3

Bit 4
Cohen Sutherland Line Clipping

```
Bit 1 | Bit 2 | Bit 3 | Bit 4
      | 1010  | 1000  | 1001
      | 1010  | 1000  | 1001
      | 0010  | 0000  | 0001
      | 0110  | 0100  | 0101
```

Points:
- P5
- P6
- P7
- P8
- P9
- P10

Areas:
- 0000
Cohen Sutherland Line Clipping
Cohen Sutherland Line Clipping
Cohen Sutherland Line Clipping

The image illustrates the Cohen Sutherland clipping algorithm, a method used in computer graphics to clip line segments to a rectangular viewport. The algorithm uses a clipping rectangle, defined by the bits 1, 2, 3, and 4, to determine which line segments to clip.

The diagram shows the clipping rectangle and some example points labeled as $P_3$, $P_4$, $P_5$, $P_6$, $P_7$, $P_8$, $P_9$, and $P_{10}$. The points are used to demonstrate how the algorithm decides which line segments are inside, outside, or on the boundary of the clipping rectangle.
Cohen Sutherland Line Clipping

Bit 1

Bit 2

Bit 3

Bit 4

P_3, P_4, P_6, P_5', P_7, P_8, P_9, P_{10}

0000, 0010, 0100, 0110

0001, 0101, 1000, 1001
Cohen Sutherland Line Clipping
Cohen Sutherland Line Clipping
Cohen Sutherland Line Clipping
Cohen Sutherland Line Clipping
Cohen Sutherland Line Clipping
Cohen Sutherland Line Clipping

Bit 1

Bit 2

Bit 3

Bit 4

P_3

P_4

P_6

P_5

P_7

P_8

P_9

P_{10}
Cohen Sutherland Line Clipping
Cohen Sutherland Line Clipping

Bit 1  Bit 2  Bit 3  Bit 4

P_3  0000  P_4  P'_7  P'_8  P_10

P'_5  0010  P_9
Cohen Sutherland Line Clipping

Bit 1 | Bit 2 | Bit 3 | Bit 4
--- | --- | --- | ---
1010 | 0010 | 0100 | 1000
1001 | 0001 | 0101 | 0101

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

Bit 1  Bit 2  Bit 3  Bit 4

0000  0001  0010  0101

1000  1001  1010  1100

P_3  P_4  P_5  P_6  P_7  P_8
Denote $p(t) = p_0 + (p_1 - p_0)t \quad t \in [0..1]$

Let $q_i$ be a point on the edge $L_i$ with outside pointing normal $N_i$.

$V(t) = p(t) - q_i$ is a parameterized vector from $q_i$ to the segment $p(t)$. 

Intersecting An Edge and a Line
Finding the Parameter $t$ of the Intersection Point

$0 = N_i \cdot V(t)$

$= N_i \cdot (p(t)-q_i)$

$= N_i \cdot (p_0+(p_1-p_0)t-q_i)$

$= N_i \cdot (p_0-q_i) + N_i \cdot (p_1-p_0)t$

$$t_i = \frac{N_i \cdot (p_0 - q_i)}{-N_i \cdot (p_1 - p_0)} = \frac{N_i \cdot (p_0 - q_i)}{-N_i \cdot \Delta}$$

where $\Delta = (p_1 - p_0)$

$t$ at intersection
Cyrus-Beck Line Clipping

- The intersection of $p(t)$ with all four edges $L_i$ is computed, resulting in up to four $t_i$ values per segment.
- If $t_i < 0$ or $t_i > 1$, $t_i$ can be discarded (why?).
- Based on the sign of $N_i \cdot \Delta$, each intersection point is classified as:
  - $PE$ (potentially entering) $< 0$
  - $PL$ (potentially leaving) $> 0$
- $PE$ with the largest $t$ and $PL$ with the smallest $t$ provide the domain of $p(t)$ inside $W$.
- If $PL$ comes before $PE$ the line is outside!
- Note: If $N_i \cdot \Delta = 0$, $t$ has no solution. However, in this case $V(t) \perp N_i$ and there is no defined intersections.
- Choice of $p1,p0$ is arbitrary!

$$\Delta = (p_1 - p_0)$$
Cyrus-Beck Line Clipping Example

Line is outside
Polygon Clipping

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

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

After Clipping
Naïve Clipping

- Find intersections
Naïve Clipping

- Find intersections
- Remove all outside vertices
Sutherland Hodgeman Clipping

- Idea: clip to window infinite boundary line one at a time
Sutherland Hodgeman Clipping
Sutherland Hodgeman Clipping
Sutherland Hodgeman Clipping
Sutherland Hodgeman Clipping
Clipping to a Boundary

- Assumes you have vertices ordered CCW!
- Loop around polygon:
  - Do inside test for each vertex in polygon sequence
  - When crossing window boundary insert new point
  - Remove points outside window boundary
Clipping to a Boundary

Window Boundary

Inside

Outside

P_1

P_2

P_3

P_4

P_5
Clipping to a Boundary

Window Boundary

Inside

Outside
Clipping to a Boundary

Window Boundary

Inside

Outside

P1, P2, P3, P4, P5
Clipping to a Boundary

Window Boundary

Inside

Outside

P_1
P_2
P_3
P_4
P_5
P'
Clipping to a Boundary

Window Boundary

Outside

Inside

P₁

P₂

P₃

P₄

P₅

P'
Clipping to a Boundary
Clipping to a Boundary

Window Boundary

P1

P2

P3

P4

P5

P'"
Clipping to a Boundary
Relation Between Successive Vertices

Assume vertex $u$ has been dealt with, vertex $v$ follows:

- $v$ added to output list
- $w$ added to output list
- no output
- $w$ and $v$ added to output list
Sutherland Hodgeman Clipping

- Assumes you have vertices ordered in List L

Initialize: $L_{in} = L$, $L_{out} = \emptyset$
For each window edge $e$ {
  $u = \text{last}(L_{in})$
  Loop over $v \in L_{in}$ {
    If $v$ is inside $e$ then
      if $u$ is outside $e$
        then $L_{out} = L_{out} \cup \text{intersect}(e, <u,v>)$
      $L_{out} = L_{out} \cup v$
    Else
      if $u$ is inside $e$
        then $L_{out} = L_{out} \cup \text{intersect}(e, <u,v>)$
      $u = v$
  }
  $L_{in} = L_{out}$
}
Notes

- Works for any convex clipping polygon
- Good for drawing – not for modeling: Overlapping edges
2D Rendering Pipeline

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

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

Let's consider a point $(wx, wy)$ in the Window and map it to $(vx, vy)$ in the Viewport. The mapping formulas are:

\[ v_x = v_{x1} + \frac{v_{x2} - v_{x1}}{w_{x2} - w_{x1}} (w_x - w_{x1}) \]

\[ v_y = v_{y1} + \frac{v_{y2} - v_{y1}}{w_{y2} - w_{y1}} (w_y - w_{y1}) \]
Summary

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
Summary of Transformations

Modeling transformation

Viewing transformations

Viewport transformation

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

3D Object Coordinates

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

2D Image Coordinates

Modeling Transformation

Viewing Transformation

Projection Transformation

Window-to-Viewport Transformation

3D World Coordinates

3D Camera Coordinates

2D Screen Coordinates