Clipping Output Primitives

- The process of removing the invisible portions of the output primitives while working with the world coordinate system (WCS)

- Clipping is necessary to avoid the "wrap-around" and "internal register overflow" problems

- Points and lines lying on the window border are considered inside.

• **Clipping** and **mapping** are the responsibility of the application programmer
**Primitives:** points, lines, polygons, text

**Point clipping:**

To determine if a point \((x, y)\) is inside a window defined by \((x_{\text{min}}, y_{\text{min}})\), lower-left corner, and \((x_{\text{max}}, y_{\text{max}})\), upper-right corner, simply test if

\[
x_{\text{min}} \leq x \leq x_{\text{max}}
\]

\[
y_{\text{min}} \leq y \leq y_{\text{max}}
\]
**Line clipping:** (Cohen-Sutherland algorithm)

- To avoid unnecessary computation, perform tests on trivially accepted cases and trivially rejected cases first.

- If both endpoints are inside the window, then the line segment is inside the window.

- If both endpoints are to the left \((x < x_{\text{min}})\), to the right \((x > x_{\text{min}})\), below \((y < y_{\text{min}})\), or above \((y > y_{\text{min}})\) the window, then the line segment is outside the window.
To perform the tests efficiently, divide the world coordinate system into 9 regions and assign each of them a four-bit code

<table>
<thead>
<tr>
<th>1001</th>
<th>1000</th>
<th>1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>bottom</td>
<td>right</td>
<td>left</td>
</tr>
</tbody>
</table>

bit 1: sign bit of \((x - x_{min})\)
bite 2: sign bit of \((x_{max} - y)\)
bite 3: sign bit of \((y - y_{min})\)
bite 4: sign bit of \((y_{max} - y)\)
The Cohen-Sutherland Algorithm

1. Compute the codes for the endpoints of the line segment to be clipped

2. Repeat until the line segment is either trivially accepted or rejected

   2.1 [Trivial Acceptance Test]
   If bitwise OR of the codes is 0000 (line segment is inside the window), draw the line segment and stop.

3. [Trivial Rejection Test]
If bitwise AND of the codes is not 0000 (line segment is outside the window), discard the line segment and stop.

4. [Subdivide the segment]

   4.1 Pick an endpoint whose code is non-zero (the endpoint that is outside the window)

   4.2 Find the first non-zero bit: this corresponds to the window edge which intersects the line segment

   4.3 Compute the intersection point and replace the outside endpoint with the intersection point
An Example

Use bit 2 of $A$ (right clipping edge) to do the subdivision
Subdivide at $C$ (Find $y$ coordinate of $C$)

$$y = m \cdot x_{\text{max}} + b$$
Example (con’t)

Use bit 4 of C (top clipping edge) to do the subdivision

Subdivide at $D$ (need to find $x$ coordinate of $D$)

$x = (y_{\text{max}} - b)/m$
Use bit 1 of B (left clipping edge) to do the subdivision

Subdivide at $E$ (need to find $y$ coordinate of $E$)

$$y = m \cdot x_{\text{min}} + b$$
Example (con’t)

Segment $ED$ is trivially accepted
Polygon clipping:

- Can not simply use a line clipper since it may generate a series of unconnected line segments

Before clipping

After clipping

- A polygon clipper should generate one or more closed areas

Before clipping

After clipping
Sutherland-Hodgman Algorithm

- Polygon boundary is clipped as a whole against the four edges of the window separately

- For each bounding edge of the window, traverse (directed) edges of the polygon and output vertices according to the following rules:

  - $v_2$ is output
  - $v_1$ is output
  - No output
  - $i$ and $v_2$ are output
An Example
(clipping against the right edge of the window)

Start with $v_0v_1$
Output: $v_1$

Process $v_1v_2$
Output: $v_1v_1'$

Process $v_2v_3$
Output: $v_1v_1'v_2'v_3$

Process $v_3v_4$
Output: $v_1v_1'v_2'v_3v_3'$
Example (con’t)

Result: $v_1 v_1' v_2' v_3 v_3'$

Process $v_4 v_5$
Output: $v_1 v_1' v_2' v_3 v_3'$

Process $v_5 v_0$
Output: $v_1 v_1' v_2' v_3 v_3' v_5' v_0$

Result: $v_1 v_1' v_2' v_3 v_3' v_5' v_0$
Disadvantage of S-H algorithm:

- Output is always a connected area

Remedy: using Weiler-Atherton’s approach

For clockwise processing of polygon vertices in S-H clipping algorithm:

- For an outside-to-inside pair of vertices, follow the polygon boundary
- For an inside-to-outside pair of vertices, follow the window boundary in a clockwise direction
Text clipping:

Usually clip the bounding box (rectangle) of an individual character or the entire string

- All-or-none character-clipping

Before clipping

\[
\begin{array}{c}
C \$ 3 3 5 \\
\end{array}
\]

After clipping

\[
\begin{array}{c}
3 3 5 \\
\end{array}
\]

- All-or-none string-clipping

Before clipping

\[
\begin{array}{c}
C \$ 3 3 5 \\
\end{array}
\]

After clipping

\[
\begin{array}{c}
\_\_\_\_ \\
\end{array}
\]