Clipping Lines

Dr. Scott Schaefer
Why Clip?

- We do not want to waste time drawing objects that are outside of viewing window (or clipping window)
Clipping Points

- Given a point \((x, y)\) and clipping window \((x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}})\), determine if the point should be drawn

\((x_{\text{max}}, y_{\text{max}})\)

\((x, y)\)

\((x_{\text{min}}, y_{\text{min}})\)
Clipping Points

- Given a point \((x, y)\) and clipping window \((x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}})\), determine if the point should be drawn

\[ x_{\text{min}} \leq x \leq x_{\text{max}} ? \]
\[ y_{\text{min}} \leq y \leq y_{\text{max}} ? \]
Clipping Points

- Given a point \((x, y)\) and clipping window \((x_{min}, y_{min}), (x_{max}, y_{max})\), determine if the point should be drawn

\[
\begin{align*}
&\quad x_{min} \leq x \leq x_{max} \\
&\quad y_{min} \leq y \leq y_{max}
\end{align*}
\]
Clipping Points

- Given a point \((x, y)\) and clipping window \((x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}})\), determine if the point should be drawn

\[
\begin{align*}
(x_{\text{min}}, y_{\text{min}}) & \leq (x_1, y_1) & \leq (x_{\text{max}}, y_{\text{max}}) \\
(x_{\text{min}}, y_{\text{min}}) & \leq (x_2, y_2) & \leq (x_{\text{max}}, y_{\text{max}})
\end{align*}
\]
Clipping Points

- Given a point \((x, y)\) and clipping window \((x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}})\), determine if the point should be drawn.

\[
\begin{align*}
&x_{\text{min}} \leq x \leq x_{\text{max}} \\
y_{\text{min}} \leq y \leq y_{\text{max}}
\end{align*}
\]

No

Yes
Clipping Lines
Clipping Lines
Clipping Lines

- Given a line with end-points \((x_0, y_0), (x_1, y_1)\) and clipping window \((x_{\text{min}}, y_{\text{min}}), (x_{\text{max}}, y_{\text{max}})\), determine if line should be drawn and clipped end-points of line to draw.

![Diagram of line clipping](attachment:image.png)
Clipping Lines
Clipping Lines – Simple Algorithm

- If both end-points inside rectangle, draw line
- Otherwise,
  intersect line with all edges of rectangle
  clip that point and repeat test
Clipping Lines – Simple Algorithm

$$(x_0, y_0)$$

$$(x_1, y_1)$$

$$(x_{\text{max}}, y_{\text{max}})$$

$$(x_{\text{min}}, y_{\text{min}})$$
Clipping Lines – Simple Algorithm

\[
\begin{aligned}
(x_0, y_0) & \quad (x_1, y_1) & \quad (x_{\text{max}}, y_{\text{max}})
\end{aligned}
\]

\[
\begin{aligned}
(x_{\text{min}}, y_{\text{min}})
\end{aligned}
\]
Clipping Lines – Simple Algorithm

\[
(\hat{x}_0, \hat{y}_0) \quad \quad \quad \quad \quad (x_{\text{max}}, y_{\text{max}}) \\
(x_{\text{min}}, y_{\text{min}}) 
\]
Clipping Lines – Simple Algorithm

\[(x_0, y_0)\]

\[(x_1, y_1)\]

\[(x_{\text{max}}, y_{\text{max}})\]

\[(x_{\text{min}}, y_{\text{min}})\]
Clipping Lines – Simple Algorithm

\[
\begin{align*}
(x_0, y_0) & \quad (x_1, y_1) \\
(x_{\min}, y_{\min}) & \quad (x_{\max}, y_{\max})
\end{align*}
\]
Clipping Lines – Simple Algorithm

\[(x_{0}, y_{0}) \rightarrow (x_{1}, y_{1}) \rightarrow (x_{\text{max}}, y_{\text{max}}) \rightarrow (x_{\text{min}}, y_{\text{min}})\]
Clipping Lines – Simple Algorithm

\[(x_0, y_0)\]

\[(x_{\text{min}}, y_{\text{min}})\]

\[(x_{\text{max}}, y_{\text{max}})\]

\[(x_1, y_1)\]
Clipping Lines – Simple Algorithm

\[(x_{\text{max}}, y_{\text{max}})\]

\[(\hat{x}_0, \hat{y}_0)\]

\[(x_{\text{min}}, y_{\text{min}})\]

\[(x_1, y_1)\]
Clipping Lines – Simple Algorithm

\[(x_{0}, y_{0})\]

\[(x_{\text{min}}, y_{\text{min}})\]

\[(x_{1}, y_{1})\]

\[(x_{\text{max}}, y_{\text{max}})\]
Clipping Lines – Simple Algorithm

\[ (x_{\min}, y_{\min}) \]

\[ (\hat{x}_0, \hat{y}_0) \]

\[ (\hat{x}_1, \hat{y}_1) \]

\[ (x_{\max}, y_{\max}) \]
Clipping Lines – Simple Algorithm

\[
\begin{align*}
(\hat{x}_0, \hat{y}_0) & \quad (x_{\text{max}}, y_{\text{max}}) \\
(x_{\text{min}}, y_{\text{min}}) & \quad (\hat{x}_1, \hat{y}_1)
\end{align*}
\]
Window Intersection

\((x_1, y_1), (x_2, y_2)\) intersect with vertical edge at \(x_{\text{right}}\)

\[ y_{\text{intersect}} = y_1 + m(x_{\text{right}} - x_1) \]
where \(m = (y_2 - y_1)/(x_2 - x_1)\)

\((x_1, y_1), (x_2, y_2)\) intersect with horizontal edge at \(y_{\text{bottom}}\)

\[ x_{\text{intersect}} = x_1 + (y_{\text{bottom}} - y_1)/m \]
where \(m = (y_2 - y_1)/(x_2 - x_1)\)
Clipping Lines – Simple Algorithm

- Lots of intersection tests makes algorithm expensive
- Complicated tests to determine if intersecting rectangle

- Is there a better way?
Trivial Accepts

- Big Optimization: trivial accepts/rejects
- How can we quickly decide whether line segment is entirely inside window
- Answer: test both endpoints
Trivial Accepts

- Big Optimization: trivial accepts/rejects
- How can we quickly decide whether line segment is entirely inside window
- Answer: test both endpoints
Trivial Rejects

- How can we know a line is outside of the window
- Answer: both endpoints on wrong side of same edge, can trivially reject the line
Trivial Rejects

- How can we know a line is outside of the window
- Answer: both endpoints on wrong side of same edge, can trivially reject the line
Cohen-Sutherland Algorithm

- Classify \( p_0, p_1 \) using region codes \( c_0, c_1 \)
- If \( c_0 \land c_1 \neq 0 \), trivially reject
- If \( c_0 \lor c_1 = 0 \), trivially accept
- Otherwise reduce to trivial cases by splitting into two segments
Cohen-Sutherland Algorithm

- Every end point is assigned to a four-digit binary value, i.e. Region code.
- Each bit position indicates whether the point is inside or outside of a specific window edge.

<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>
## Cohen-Sutherland Algorithm

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

The diagram shows a 4-quadrant clipping area with boundary codes for clipping.
Cohen-Sutherland Algorithm

- Classify $p_0, p_1$ using region codes $c_0, c_1$
- If $c_0 \land c_1 \neq 0$, trivially reject
- If $c_0 \lor c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments
Cohen-Sutherland Algorithm

- Classify $p_0, p_1$ using region codes $c_0, c_1$
- If $c_0 \land c_1 \neq 0$, trivially reject
- If $c_0 \lor c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments
Cohen-Sutherland Algorithm

- Classify $p_0, p_1$ using region codes $c_0, c_1$
- If $c_0 \land c_1 \neq 0$, trivially reject
- If $c_0 \lor c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments

Line is outside the window! reject
Cohen-Sutherland Algorithm

- Classify \( p_0, p_1 \) using region codes \( c_0, c_1 \)
- If \( c_0 \land c_1 \neq 0 \), trivially reject
- If \( c_0 \lor c_1 = 0 \), trivially accept
- Otherwise reduce to trivial cases by splitting into two segments

Line is inside the window! draw
Cohen-Sutherland Algorithm

- Classify $p_0$, $p_1$ using region codes $c_0$, $c_1$
- If $c_0 \land c_1 \neq 0$, trivially reject
- If $c_0 \lor c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments
Cohen-Sutherland Algorithm

<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>
Cohen-Sutherland Algorithm

<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>

![Diagram of Cohen-Sutherland Algorithm](image-url)
Cohen-Sutherland Algorithm
Cohen-Sutherland Algorithm
Cohen-Sutherland Algorithm

```
1001

0001

0101

1000

0000

0100

1010

0010

0110
```
Cohen-Sutherland Algorithm
Cohen-Sutherland Algorithm

```
1001
1000
1010
0001
0000
0010
0101
0100
0110
```
Cohen-Sutherland Algorithm

<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>
Cohen-Sutherland Algorithm

<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>

A line segment with coordinates 0001-0000 and 0000-0100 is represented by a blue line.
Cohen-Sutherland Algorithm

The diagram illustrates the Cohen-Sutherland line clipping algorithm, which is used in computer graphics for determining whether a line segment is completely inside, completely outside, or partially inside a rectangular clipping area.

The algorithm uses 4-bit codes for each endpoint of the line, and the clipping area is divided into 9 regions. Each region is defined by a code that indicates its location relative to the clipping area:

- **0000**: Inside
- **0001**: Right
- **0010**: Above
- **0011**: Right and above
- **0100**: Left
- **0101**: Below
- **0110**: Left and below
- **1000**: Right and below
- **1001**: Left and above
- **1111**: Outside

By comparing the codes of the endpoints, the algorithm determines the clipping status of the line and clips it accordingly.
Cohen-Sutherland Algorithm
Cohen-Sutherland Algorithm

```
1001 1000 1010
```

```
0001 0000 0010
```

```
0101 0100 0110
```
Cohen-Sutherland Algorithm

\[
\begin{array}{|c|c|c|}
\hline
1001 & 1000 & 1010 \\
\hline
0001 & 0000 & 0010 \\
\hline
0101 & 0100 & 0110 \\
\hline
\end{array}
\]
Cohen-Sutherland Algorithm
Cohen-Sutherland Algorithm

1001 | 1000 | 1010
---|---|---
0001 | 0000 | 0010
0101 | 0100 | 0110
Liang-Barsky Algorithm

- Uses parametric form of line for clipping
- Lines are oriented
  Classify lines as moving inside to out or outside to in
- Don’t find actual intersection points
  Find parameter values on line to draw
Liang-Barsky Algorithm

- Initialize interval to \([t_{min}, t_{max}] = [0, 1]\)
- For each boundary
  
  Find parametric intersection \(t\) with boundary

  If moving in to out, \(t_{max} = \min(t_{max}, t)\)
  
  Else \(t_{min} = \max(t_{min}, t)\)

  If \(t_{min} > t_{max}\), reject line
Intersecting Two Parametric Lines

\[(x_3, y_3)\]
\[(x_1, y_1)\]
\[(x_0, y_0)\]
\[(x_2, y_2)\]
Intersecting Two Parametric Lines

\[ x(t) = x_0 + (x_1 - x_0)t \]
\[ y(t) = y_0 + (y_1 - y_0)t \]
\[ 0 \leq t \leq 1 \]
Intersecting Two Parametric Lines

\[ x(t) = x_0 + (x_1 - x_0)t \]
\[ y(t) = y_0 + (y_1 - y_0)t \]

\[ x(0) = x_0 \]
\[ y(0) = y_0 \]
\[ x(1) = x_1 \]
\[ y(1) = y_1 \]
Intersecting Two Parametric Lines

\[ x_0 + (x_1 - x_0)t = x_2 + (x_3 - x_2)s \]
\[ y_0 + (y_1 - y_0)t = y_2 + (y_3 - y_2)s \]
Intersecting Two Parametric Lines

\[
\begin{pmatrix}
  x_1 - x_0 & x_2 - x_3 \\
  y_1 - y_0 & y_2 - y_3
\end{pmatrix}
\begin{pmatrix}
  t \\
  s
\end{pmatrix}
= \begin{pmatrix}
  x_2 - x_0 \\
  y_2 - y_0
\end{pmatrix}
\]

\[
(x_0, y_0) \quad (x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3)
\]
Intersecting Two Parametric Lines

\[
\begin{bmatrix}
  x_1 - x_0 & x_2 - x_3 \\
  y_1 - y_0 & y_2 - y_3
\end{bmatrix}
\begin{bmatrix}
t \\
s
\end{bmatrix}
= 
\begin{bmatrix}
x_2 - x_0 \\
y_2 - y_0
\end{bmatrix}
\]

Substitute \( t \) or \( s \) back into equation to find intersection
Liang-Barsky: Classifying Lines

\[ p(t) \]

\[ p(1) \]

\[ p(0) \]

\[ x_1 < x_0 \quad \text{moving out} \]
Liang-Barsky: Classifying Lines

\[ p(t) \]

\[ x_0 < x_1 \quad \text{moving in} \]
Liang-Barsky: Classifying Lines

\[ p(t) \]

\[ \begin{align*}
  p(0) \\
  p(1)
\end{align*} \]

\[ x_1 < x_0 \quad \text{moving in} \]
Liang-Barsky: Classifying Lines

\[ p(t) \]

\[ p(0) \]

\[ p(1) \]

\[ x_0 < x_1 \quad \text{moving out} \]
Liang-Barsky: Classifying Lines

\[ y_1 < y_0 \quad \text{moving in} \]
Liang-Barsky: Classifying Lines

\[ y_0 < y_1 \quad \text{moving out} \]
Liang-Barsky: Classifying Lines

\[ y_1 < y_0 \quad \text{moving out} \]
Liang-Barsky: Classifying Lines

\[ p(0) \rightarrow p(t) \rightarrow p(1) \]

\[ y_0 < y_1 \quad \text{moving in} \]
Liang-Barsky Algorithm

\[ p(t) \]

\[ p(0) \]

\[ p(1) \]
Liang-Barsky Algorithm
Liang-Barsky Algorithm
Liang-Barsky Algorithm

\[ p(t) \]

\[ p(0) \]

\[ p(0.8) \]
Liang-Barsky Algorithm
Liang-Barsky Algorithm
Liang-Barsky Algorithm
Liang-Barsky Algorithm

$p(0)$

$p(1)$

$p(\cdot.7)$
Liang-Barsky Algorithm

\[ p(0) \rightarrow p(1) \]
Liang-Barsky Algorithm

\[ p(1) \]

\[ p(.2) \]
Liang-Barsky Algorithm
Liang-Barsky Algorithm
Liang-Barsky Algorithm

$p(.1)$

$p(1)$
Liang-Barsky Algorithm
Liang-Barsky Algorithm
Liang-Barsky Algorithm
Liang-Barsky Algorithm
Liang-Barsky Algorithm

$p(0)$

$p(1)$
Liang-Barsky Algorithm

\[ p(0) \]

\[ p(1) \]
Liang-Barsky Algorithm
Liang-Barsky Algorithm

$p(0)$

$p(.6)$
Liang-Barsky Algorithm

$p(0)$

$p(.6)$
Liang-Barsky Algorithm
Liang-Barsky Algorithm
Comparison

- **Cohen-Sutherland**
  - Repeated clipping is expensive
  - Best used when trivial acceptance and rejection is possible for most lines

- **Liang-Barsky**
  - Computation of \( t \)-intersections is cheap (only one division)
  - Computation of \((x,y)\) clip points is only done once
  - Algorithm doesn’t consider trivial accepts/rejects
  - Best when many lines must be clipped
Line Clipping – Considerations

- Just clipping end-points does not produce the correct results inside the window
- Must also update $sum$ in midpoint algorithm

- Clipping against non-rectangular polygons is also possible but seldom used