Line Clipping

R. J. Renka

Department of Computer Science & Engineering
University of North Texas

03/01/2016
The clipping problem is that of finding the portion of a geometric primitive (point, line, or polygon) that lies inside a view volume. For points, it is just a containment test; lines require computing intersections of line segments with view volume boundaries; and polygons, defined by sequences of vertices, involve additional complexity associated with keeping track of the order of the vertices.

2-D Line-clipping Problem: Given a line segment defined by $p_0$ and $p_1$ in $\mathbb{R}^2$, and a rectangular clip window $[X_{\min}, X_{\max}] \times [Y_{\min}, Y_{\max}]$, clip the line segment against the window; i.e., replace the line segment by its intersection with the window.

The general problem of finding the intersection of a pair of line segments is not trivial. It is much simpler to intersect a line segment with a horizontal or vertical line defined by constant $y$ or constant $x$.  

R. J. Renka  
Line Clipping
Cohen-Sutherland Line Clipping Algorithm

We partition the plane into nine regions by extending the four clip window boundaries to infinity in both directions, and then assign 4-bit region codes or outcodes to the regions by arbitrarily ordering the boundaries. The diagram on the following page is associated with the checking order TBRL, corresponding to Top, Bottom, Right, Left. We assign a 1-bit where the region is strictly outside the boundary (in the half-plane not containing the window), and a 0-bit where the region is on the same side as the window. Thus, only the window itself is assigned all zeros. Since the high-order bit is associated with the top boundary for example, only the three regions above the window (outside the top boundary) have high-order bit equal to 1.

\[
\begin{align*}
\text{Top} & \quad b_3 = \text{sign}(Y_{\max} - y) \\
\text{Bottom} & \quad b_2 = \text{sign}(y - Y_{\min}) \\
\text{Right} & \quad b_1 = \text{sign}(X_{\max} - x) \\
\text{Left} & \quad b_0 = \text{sign}(x - X_{\min})
\end{align*}
\]
Cohen-Sutherland Line Clipping Example

The diagram illustrates the process of line clipping using the Cohen-Sutherland algorithm. The line segment $p_0 - p_1$ is clipped against the rectangular clipping window defined by the coordinates $(X_{\text{min}}, Y_{\text{min}})$ and $(X_{\text{max}}, Y_{\text{max}})$. The points are encoded using the Cohen-Sutherland clipping code:

- $p_0$ has code $0101$
- $p_1$ has code $1010$
- $A$ has code $1001$
- $B$ has code $0001$
- $C$ has code $0010$

The clipping process proceeds as follows:

1. $p_0 - p_1$ is clipped against the window.
2. The line segment is moved to $A - p_1$.
3. The line segment is moved to $A - C$.
4. The line segment is moved to $A - B$.

The final line segment is $A - B$.
Cohen-Sutherland Line Clipping Algorithm

Assign region codes $c_0$ to $p_0$ and $c_1$ to $p_1$

while (1) {
    if ($c_0 \mid c_1 == 0$) break;
    if ($c_0 \& c_1 != 0$) {
        Set the line segment to the empty set
        break;
    }
    $c = (c_0 != 0) ? c_0 : c_1$;
    Test the bits of $c$ in left-to-right order to find which boundary is crossed, and find the intersection point
    Replace the outside point by the intersection point, and compute the new region code by zeroing the appropriate bits
}
Draw the line segment
The algorithm is efficient when region code testing is cheap and most line segments are trivially rejected due to a relatively small window, or accepted without alteration due to a relatively large window. In the example, three intersections are computed when only two are actually needed. A worst-case example requires four intersections. The Liang-Barsky parametric line clipping algorithm is more efficient when a lot of clipping is needed.

Both algorithms are easily generalized to 3-D. To extend the Cohen-Sutherland algorithm to a three-dimensional orthogonal view volume (a rectangular box), we use 6-bit region codes with additional bits for the near and far clipping planes. Then the worst-case line segment requires computation of six intersections. A frustum is also defined by six planar boundaries but, because the planes are not axis-aligned (defined by constant \(x\), \(y\), or \(z\)), region code assignment and intersection computations are more expensive.
To find the intersection with the left or right boundary, we write $y$ as the linear function of $x$ that interpolates the endpoints, and plug in $x = X_{\text{min}}$ or $x = X_{\text{max}}$. For $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$, the intersection with the left boundary is

$$(X_{\text{min}}, y_0 + m(X_{\text{min}} - x_0)),$$

where the slope is $m = (y_1 - y_0)/(x_1 - x_0)$. Note that, if $x_1 - x_0$ were 0, we would not have found an intersection with a vertical boundary. To find the intersection with the top or bottom boundary, we reverse the roles of $x$ and $y$; i.e., write $x$ as the linear function of $y$ that interpolates the endpoints, and plug in $y = Y_{\text{max}}$ or $y = Y_{\text{min}}$. 

R. J. Renka

Line Clipping
Intersection of a Line Segment with a Clipping Plane

To find the intersection of \( p_0 - p_1 \) with the plane \( z = Z_{\text{min}} \), we write \( x \) and \( y \) as interpolatory linear functions of \( z \)

\[
x = x_0 + \frac{x_1 - x_0}{z_1 - z_0} (z - z_0), \quad y = y_0 + \frac{y_1 - y_0}{z_1 - z_0} (z - z_0),
\]

and substitute \( z = Z_{\text{min}} \). As an alternative derivation, the parametric representation of the line segment is

\[
S = \{ p_0 + t(p_1 - p_0) : t \in [0, 1] \} \subset \mathbb{R}^3.
\]

Thus, \( p = (x, y, z) \in S \Rightarrow x = x_0 + t(x_1 - x_0), \)
\[
y = y_0 + t(y_1 - y_0), \text{ and } z = z_0 + t(z_1 - z_0) \text{ for } t \in [0, 1].
\]

To find \( S \cap \{(x, y, z) : z = Z_{\text{min}}\} \), assuming \( z_1 \neq z_0 \), we solve \( z = Z_{\text{min}} = z_0 + t(z_1 - z_0) \) for \( t = (Z_{\text{min}} - z_0)/(z_1 - z_0) \), which gives the same expressions derived above:

\[
x = x_0 + \frac{Z_{\text{min}} - z_0}{z_1 - z_0} (x_1 - x_0), \quad y = y_0 + \frac{Z_{\text{min}} - z_0}{z_1 - z_0} (y_1 - y_0).
\]
Liang-Barsky Line Clipping Algorithm

Parametric representation: \( \mathbf{v} = \mathbf{v}_0 + u(\mathbf{v}_1 - \mathbf{v}_0) \)

\( t_0 = 0, \ t_1 = 1, \ dx = x_1-x_0, \ dy = y_1-y_0 \)

for each boundary \( b \) in \{Left, Right, Bottom, Top\}
  - if (\( b == \text{Left} \)) \( p = -dx, \ q = -(X_{\text{min}}-x_0) \)
  - if (\( b == \text{Right} \)) \( p = dx, \ q = X_{\text{max}}-x_0 \)
  - if (\( b == \text{Bottom} \)) \( p = -dy, \ q = -(Y_{\text{min}}-y_0) \)
  - if (\( b == \text{Top} \)) \( p = dy, \ q = Y_{\text{max}}-y_0 \)

\( u = q/p \)  // Parameter value of \( b \)

if (\( p == 0 \))  // Line parallel to \( b \)
  - if (\( q < 0 \)) return false  // Reject line segment
  elseif (\( p < 0 \))  // Outside to inside
    - if (\( u > t_1 \)) return false
    - if (\( u > t_0 \)) \( t_0 = u \)  // Clip
else  // Inside to outside
Liang-Barsky Line Clipping Algorithm

if (u < t0) return false
if (u < t1) t1 = u  // Clip
end
end
x0 = x0 + t0*dx, y0 = y0 + t0*dy
x1 = x0 + t1*dx, y1 = y0 + t1*dy
return true