Line Clipping

R. J. Renka

Department of Computer Science & Engineering
University of North Texas

03/01/2016

Problem

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 $\mathbf{p}_0$ and $\mathbf{p}_1$ in $\mathbf{R}^2$, and a rectangular clip window $[X_{\text{min}}, X_{\text{max}}] \times [Y_{\text{min}}, Y_{\text{max}}]$, clip the line segment against the window; i.e., replace the line segment by its intersection with the window.

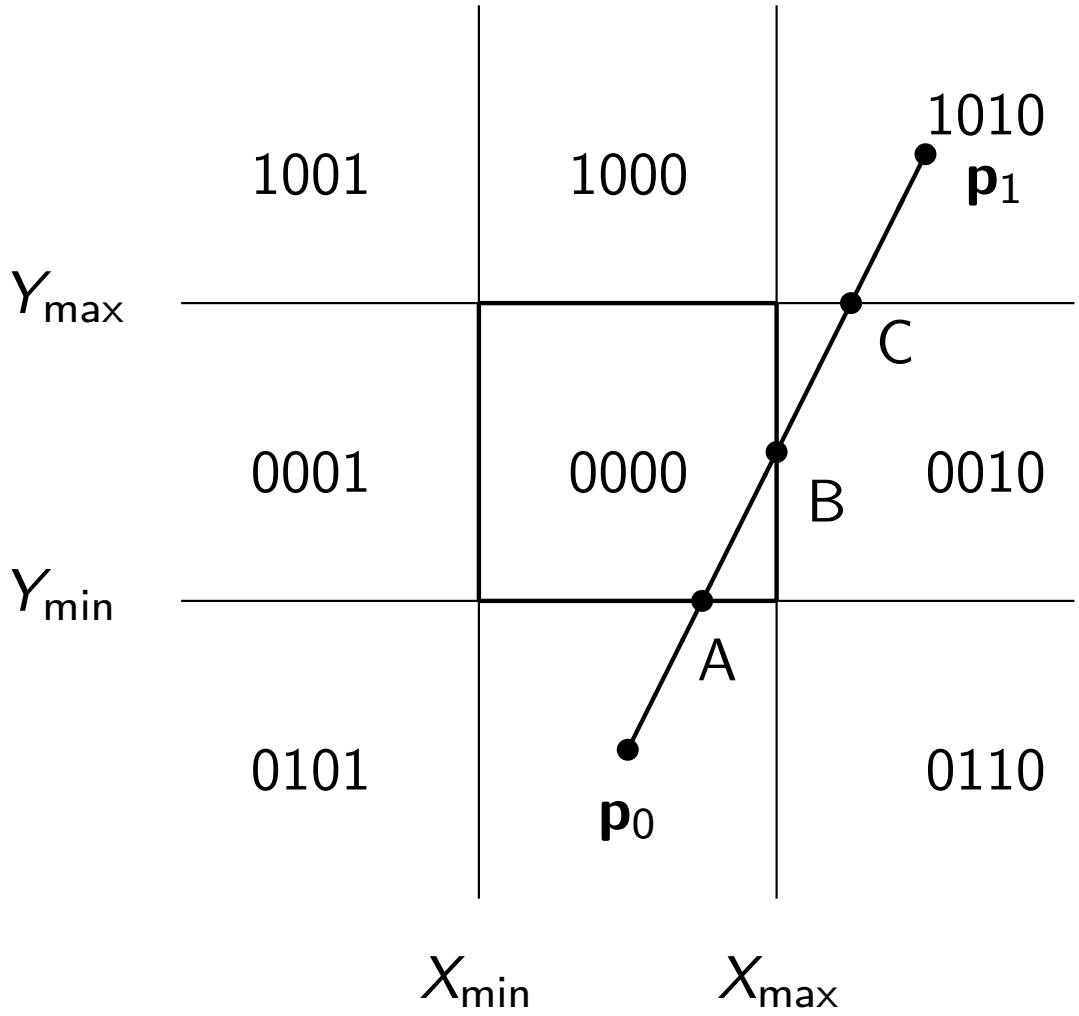
The general problem of finding the intersection of a pair of line segments is not trivial. It is much simpler to intersect a line segment with a horizontal or vertical line defined by constant y or constant x.

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.

$$\begin{aligned} \text{Top} \quad & b_3 = \text{sign}(Y_{\text{max}} - y) \\ \text{Bottom} \quad & b_2 = \text{sign}(y - Y_{\text{min}}) \\ \text{Right} \quad & b_1 = \text{sign}(X_{\text{max}} - x) \\ \text{Left} \quad & b_0 = \text{sign}(x - X_{\text{min}}) \end{aligned}$$

Cohen-Sutherland Line Clipping Example



$$\mathbf{p}_0 - \mathbf{p}_1 \longrightarrow A - \mathbf{p}_1 \longrightarrow A - C \longrightarrow A - B$$

Cohen-Sutherland Line Clipping Algorithm

Assign region codes c0 to $\mathbf{p}_0$ and c1 to $\mathbf{p}_1$

```
while (1) {
    if (c0 | c1 == 0) break;
    if (c0 & c1 != 0) {
        Set the line segment to the empty set
        break;
    }
    c = (c0 != 0) ? c0 : c1;
    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
```

Cohen-Sutherland Line Clipping Algorithm continued

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.

Computing Intersections

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 $\mathbf{p}_0 = (x_0, y_0)$ and $\mathbf{p}_1 = (x_1, y_1)$, 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 $y = Y_{\text{min}}$.

Intersection of a Line Segment with a Clipping Plane

To find the intersection of $\mathbf{p}_0 - \mathbf{p}_1$ 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

$$S = \{\mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0) : t \in [0, 1]\} \subset \mathbf{R}^3.$$

Thus, $\mathbf{p} = (x, y, z) \in S \Rightarrow x = x_0 + t(x_1 - x_0)$, $y = y_0 + t(y_1 - y_0)$, and $z = z_0 + t(z_1 - z_0)$ 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: $\mathbf{v} = \mathbf{v}_0 + u(\mathbf{v}_1 - \mathbf{v}_0)$

```
t0 = 0,  t1 = 1,  dx = x1-x0,  dy = y1-y0
for each boundary b in {Left, Right, Bottom, Top}
   if (b == Left) p = -dx,  q = -(Xmin-x0)
   if (b == Right) p = dx,  q = Xmax-x0
   if (b == Bottom) p = -dy,  q = -(Ymin-y0)
   if (b == Top) p = dy,  q = Ymax-y0
   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 > t1) return false
      if (u > t0) t0 = 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
```