Clipping Algorithms

Hans Dulimarta

1 Line Clipping Algorithms

Figure 1: 2D Line Clipping: (a) against rectangular window, (b) against convex polygon

   - clips a line against **rectangular window** area
   - uses 4-bit (6-bit) code to identify the location of line end points

2. Liang-Barsky (1984) algorithm
   - clips a line against any **convex** polygon
   - uses parametric equation of line

1.1 Cohen-Sutherland Algorithm

Regions around clipping window are labelled using 4-bit code: $b_3b_2b_1b_0 = \begin{pmatrix} T & B & R & L \end{pmatrix}$

<table>
<thead>
<tr>
<th>$b_3$</th>
<th>$b_2$</th>
<th>$b_1$</th>
<th>$b_0$</th>
<th>$T$</th>
<th>$B$</th>
<th>$R$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1000</td>
<td>1010</td>
<td>$y_{\text{max}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
<td>$y_{\text{min}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
<td>$x_{\text{min}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$x_{\text{max}}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To clip $P_1P_2$, determine the out code / region code of each end point. In Figure 1 (left) code($A$) is 0000, code ($H$) is 0010.
Let $c_1 = \text{code}(P_1)$ and $c_2 = \text{code}(P_2)$. Observe the following properties of bitwise operations on $c_1$ and $c_2$:

(a) $c_1|c_2 = 0$ both $P_1$ and $P_2$ are completely inside the rectangle.

(b) $c_1\&c_2 \neq 0$ both $P_1$ and $P_2$ are completely outside the rectangle and on one side of the window boundary (line $EF$).

(c) $c_1\&c_2 = 0$ one of the following is true
- One end point is inside and the other is outside (line $CD$)
- Both end points are outside (line $GH$ or $IJ$).

For case (c) we have to find intersection between the line and window boundaries. A '1' in the region code indicates which boundary to intersect with.

Sketch of algorithm:

```java
1 boolean clip (Point p1, Point p2)
2 {
3     c1 = code (p1);
4     c2 = code (p2);
5     while (true) {
6         if (c1 | c2 == 0)
7             return true; /* line is completely inside */
8         if (c1 & c2 != 0)
9             return false; /* line is completely outside */
10     }
11     /* find intersection (update the points and the code) */
12     if (c1 != 0) {
13         findIntersect (p1, p2, c1);
14         c1 = code (P1);
15     }
16     if (c2 != 0) {
17         findIntersect (p2, p1, c2);
18         c2 = code (P2);
19     }
20 }
```

Issue: how to handle vertical lines?
1.2 Liang-Barsky Algorithm

A faster algorithm than Cohen-Sutherland’s algorithm. It can be implemented using mostly integer arithmetic (floating-point arithmetic is needed only in the final step). Liang-Barsky algorithm is an improvement of Cyrus-Beck algorithm (1978).

1.2.1 General Idea

Given parametric equation of a line $P_1P_2$, with parameter $s$:

$$p(s) = p_1 + s(p_2 - p_1)$$

- “Squeeze” the line from both endpoints
- Find $s$ that corresponds to intersection of the line with the clipping window boundaries. If $0 \leq s \leq 1$, the intersection is inside the line. Otherwise (if $s < 0$ or $s > 1$), the intersection is outside the line and can be ignored.
- Classify intersection point as ENTER or LEAVE. View the line as a vector (direction is important)
- To find the clipped line segment, start with $s_E = 0$ and $s_L = 1$, then
  1. Update $s_E$ to the next larger ENTERing intersection
  2. Update $s_L$ to the next smaller LEAVEing intersection

If during the update $s_E > s_L$, reject the line. Why?
1.2.2 How to Identify Enter or Leave

The expression $\mathbf{n} \cdot \mathbf{d}$ can be used for boundaries at any orientation. For horizontal/vertical boundaries, classification of "Enter" or "Leave" can be simplified.

For intersection with left/right boundary $\mathbf{n} = \begin{bmatrix} \pm 1 \\ 0 \end{bmatrix}$ or top/bottom boundary $\mathbf{n} = \begin{bmatrix} 0 \\ \pm 1 \end{bmatrix}$

- Left boundary: $\mathbf{n} \cdot \mathbf{d} = -(x_2 - x_1)$
- Right boundary: $\mathbf{n} \cdot \mathbf{d} = (x_2 - x_1)$
- Top boundary: $\mathbf{n} \cdot \mathbf{d} = (y_2 - y_1)$
- Bottom boundary: $\mathbf{n} \cdot \mathbf{d} = -(y_2 - y_1)$

1.2.3 How to Calculate $s$

Parametric equation of $P_1P_2$:

$$\mathbf{p}(s) = \mathbf{p}_1 + s(\mathbf{p}_2 - \mathbf{p}_1)$$

or

$$x = x_1 + s(x_2 - x_1)$$
$$y = y_1 + s(y_2 - y_1)$$

For an intersection point $(x_a, y_a)$:

$$x_a = x_1 + s_a(x_2 - x_1)$$
$$y_a = y_1 + s_a(y_2 - y_1)$$

The intersection with the left boundary is $(x_{\text{min}}, y_L)$, the $s$ value is calculated from:

$$s_{\text{left}} = \frac{x_{\text{min}} - x_1}{x_2 - x_1} = \frac{x_1 - x_{\text{min}}}{-(x_1 - x_2)}$$

Similar technique can be used for calculating the intersection points with the other three boundaries.
Notice the following:

- The denominators in the above formulae match the expression for identifying Enter or Leave.
- Both numerator and denominator are integer expressions

\[
s_{\text{left}} = \frac{x_1 - x_{\text{min}}}{-(x_2 - x_1)} \quad s_{\text{right}} = \frac{x_{\text{max}} - x_1}{x_2 - x_1}
\]

\[
s_{\text{bottom}} = \frac{y_1 - y_{\text{min}}}{-(y_2 - y_1)} \quad s_{\text{top}} = \frac{y_{\text{max}} - y_1}{y_2 - y_1}
\]

In general \( s = \frac{\text{num}}{\text{denum}} \), with the following three possibilities:

- \( \text{denum} > 0 \): Leave type of intersection,
- \( \text{denum} < 0 \): Enter type of intersection,
- \( \text{denum} = 0 \land \text{num} < 0 \): Line is outside the window boundary.
Implementation of algorithm (for rectangular clipping window):

```c
boolean clip2D (Point P1, Point P2) {
    x1 = P1.x; y1 = P1.y;
    x2 = P2.x; y2 = P2.y;
    sE = 0; sL = 1;
    dx = x2 - x1; dy = y2 - y1;
    if (clip (x1 - xmin, -dx, sE, sL)) { /* left */
        if (clip (xmax - x1, dx, sE, sL)) /* right */
            if (clip (y1 - ymin, -dy, sE, sL)) /* bottom */
                if (clip (ymax - y1, dy, sE, sL)) /* top */
                    {
                        if (sL < 1) {
                            /* P2 has moved */
                            x2 = x1 + sL * dx;
                            y2 = y1 + sL * dy;
                        }
                        if (sE > 0) {
                            /* P1 has moved */
                            x1 = x1 + sE * dx;
                            y1 = y1 + sE * dy;
                        }
                        return true;
                    }
        return false;
    }

    /* the following function returns true */
    /* if the line is accepted */
    boolean clip (int num, int denum, float& aE, float& aL) {
        float s;
        if (denum < 0) { /* "Enter" type */
            s = num / denum; /* new sE */
            if (s > aL) return false; /* cross over !!! */
            else if (s > aE) aE = s; /* update aE to a higher s */
        }
        else if (denum > 0) { /* "Leave" type */
            s = num / denum; /* new sL */
            if (s < aE) return false; /* cross over !!! */
            else if (s < aL) aL = s; /* update aL to a lower s */
        }
        else /* perpendicular */
            return num >= 0;
        return true;
    }
}```
### 1.3 Clipping Against Convex Polygons

Intersection between $PQ$ and the polygon $ABCDE$.  

<table>
<thead>
<tr>
<th>Intersection With</th>
<th>Intersection Point (Enter/Leave)</th>
<th>$s$</th>
<th>$s_E$</th>
<th>$s_L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Intersection between $KL$ and the polygon $ABCDE$.  

<table>
<thead>
<tr>
<th>Intersection With</th>
<th>Intersection Point (Enter/Leave)</th>
<th>$s$</th>
<th>$s_E$</th>
<th>$s_L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1.4 Clipping a Line in 3D

1.4.1 Cohen-Sutherland Algorithm

Cohen-Sutherland algorithm can be extended to 3D and the following 6-bit region code is used:

\[ b_5b_4b_3b_2b_1b_0 = \begin{array}{c|c|c|c|c|c}
\text{Fr} & \text{Bk} & \text{T} & \text{B} & \text{R} & \text{L}
\end{array} \]

Two additional "Front" and "Back" surfaces are on the Z-axis.

1.4.2 Liang-Barsky Algorithm

Extension of Liang-Barsky algorithm for 3D clipping:

- Parametric line equations in 3D have the same format as in 2D:
  \[ p(s) = p_1 + s(p_2 - p_1) \]
  or
  \[
  \begin{align*}
  x(s) &= x_1 + s(x_2 - x_1) \\
  y(s) &= y_1 + s(y_2 - y_1) \\
  z(s) &= z_1 + s(z_2 - z_1)
  \end{align*}
  \]

- Intersection between two lines in 2D is now intersection between a line and a plane in 3D.

\[ P_0 \] is any point on the plane, \( R \) is intersection of \( P_1P_2 \) and the plane, \( n \) is the normal vector. Must find \( s \) such that \( p(s) = R \).

\[
\begin{align*}
\overrightarrow{P_0R} \cdot n &= 0 \\
[ R - p_0 ] \cdot n &= 0 \\
[ p(s) - P_0 ] \cdot n &= 0 \\
[ p_1 + s(p_2 - p_1) - p_0 ] \cdot n &= 0 \\
[ s(p_2 - p_1) - (p_0 - p_1) ] \cdot n &= 0 \\
s(p_2 - p_1) \cdot n - (p_0 - p_1) \cdot n &= 0
\end{align*}
\]

\[ s = \frac{(p_0 - p_1) \cdot n}{(p_2 - p_1) \cdot n} \]

For standard clipping volume (a box who opposite corner is at \((x_{\text{min}}, y_{\text{min}}, z_{\text{min}})\) and \((x_{\text{max}}, y_{\text{max}}, z_{\text{max}})\)) the normal vector \( n \) is one of the following six vectors

\[
\begin{bmatrix}
\pm 1 \\
0 \\
0
\end{bmatrix}, \begin{bmatrix}
0 \\
\pm 1 \\
0
\end{bmatrix}, \begin{bmatrix}
0 \\
0 \\
\pm 1
\end{bmatrix}
\]

and \( s \) can be calculated from

\[
\begin{align*}
& s = \frac{x_{\text{max}} - x_1}{x_2 - x_1} \\
& s = \frac{y_{\text{max}} - y_1}{y_2 - y_1} \\
& s = \frac{z_{\text{max}} - z_1}{z_2 - z_1} \\
& s = \frac{x_{\text{min}}}{- (x_2 - x_1)} \\
& s = \frac{y_{\text{min}}}{- (y_2 - y_1)} \\
& s = \frac{z_{\text{min}}}{- (z_2 - z_1)}
\end{align*}
\]
2 Polygon Clipping

Goal: clip a polygon against either a rectangular window area or area of any shape.

- Clipped polygon may be either convex or concave.
- Clipping polygon is usually convex.

We cannot simply clip lines because clipping a concave polygon may generate > 1 polygon.

2.1 Sutherland-Hodgman Algorithm (1974)

- Divide and conquer strategy. Decompose into simple problem: clip a polygon against a single infinite line
- The algorithm can be used for clipping a polygon (convex / concave) against any convex polygon
- Extendable to 3D (clipping a volume against any convex volume)

General Idea

\[
\text{input vertices} \rightarrow \text{clip against 1st boundary} \rightarrow \text{clip against 2nd boundary} \rightarrow \ldots \rightarrow \text{clip against n\textsuperscript{th} boundary} \rightarrow \text{clipped output}
\]

For a rectangular clipping window, a 4-stage clipper is used:

\[
\text{input vertices} \rightarrow \text{left clipper} \rightarrow \text{right clipper} \rightarrow \text{bottom clipper} \rightarrow \text{left clipper} \rightarrow \text{vertices of clipped polygon}
\]

Clipping Process

- Visit all vertices of the (to be) clipped polygon \(v_1, v_2, \ldots, v_n\) and back to \(v_1\) in CCW order.
- While visiting these vertices, observe whether each vertex is inside or outside the current clipping polygon
- At each step examine the relationship between two successive vertices of the clipped polygon with the current sid of the clipping polygon and 0, 1, or 2 vertices are added to the output list that defines the clipped polygon.
- Order of vertices (of the clipped polygon) in the output list is important!
- The order of how each side of the clipping polygon is used is not important!
Four possible relationships between the clipped vertices and clipping boundary:

1. **in → in**
   - Output: Q

2. **in → out**
   - Output: R

3. **out → out**
   - Output: none

4. **out → in**
   - Output: S, Q (in that order!!)

**Testing Inside/Outside**

- \( n \cdot \vec{SP} < 0 \) : P is inside
- \( n \cdot \vec{SQ} > 0 \) : Q is outside
2.2 Example

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Side</th>
<th>Against AB</th>
<th>Vertex</th>
<th>Side</th>
<th>Against BC</th>
<th>Vertex</th>
<th>Side</th>
<th>Against CD</th>
<th>Vertex</th>
<th>Side</th>
<th>Against DE</th>
<th>Vertex</th>
<th>Side</th>
<th>Against EA</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>PQ</td>
<td></td>
<td>Q</td>
<td>QR</td>
<td></td>
<td>R</td>
<td>RS</td>
<td></td>
<td>S</td>
<td>ST</td>
<td></td>
<td>T</td>
<td>TP</td>
<td></td>
</tr>
</tbody>
</table>