Introduction to Computer Graphics

(C S 6 0 2)

Lecture 14

Clipping-I

14.1 Concept

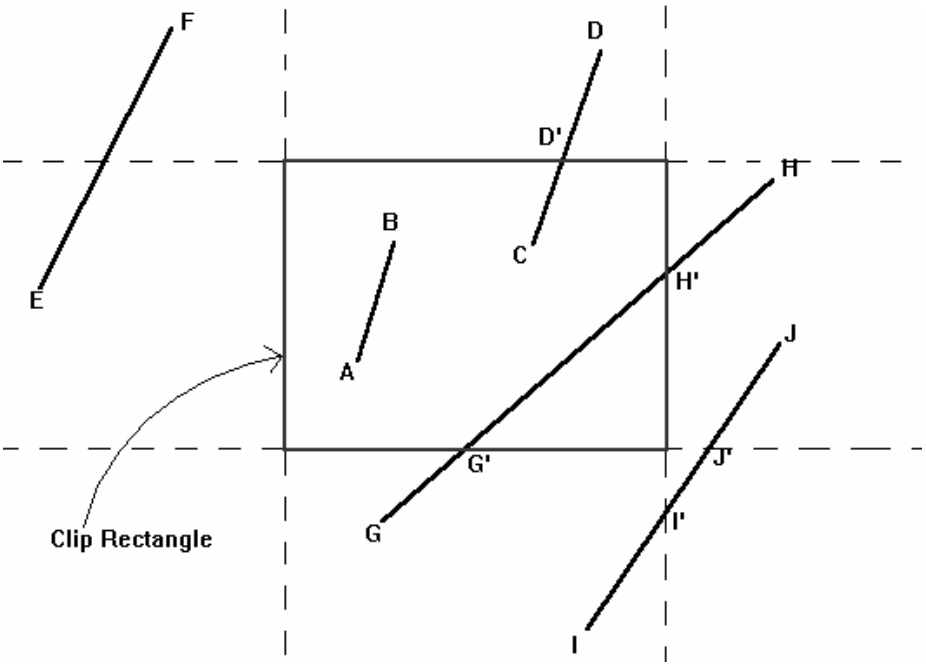
It is desirable to restrict the effect of graphics primitives to a sub-region of the canvas, to protect other portions of the canvas. All primitives are clipped to the boundaries of this **clipping rectangle**; that is, primitives lying outside the clip rectangle are not drawn.

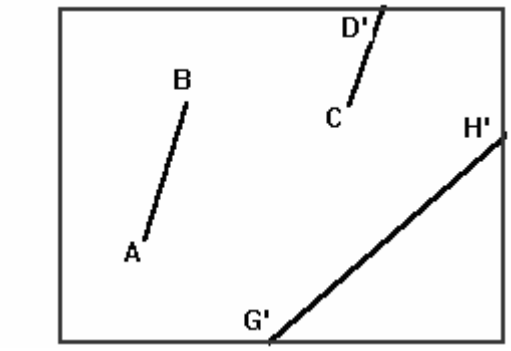
The default clipping rectangle is the full canvas (the screen), and it is obvious that we cannot see any graphics primitives outside the screen.

A simple example of line clipping can illustrate this idea:

This is a simple example of line clipping: the display window is the canvas and also the default clipping rectangle, thus all line segments inside the canvas are drawn.

The red box is the clipping rectangle we will use later, and the dotted line is the extension of the four edges of the clipping rectangle.





a) Point Clipping

Assuming a rectangular clip window, point clipping is easy. we save the point if:

$x_{min} <= x <= x_{max}$
$y_{min} <= y <= y_{max}$

b) Line Clipping

This section treats clipping of lines against rectangles. Although there are specialized algorithms for rectangle and polygon clipping, it is important to note that other graphic primitives can be clipped by repeated application of the line clipper.

c) Clipping Individual Points

Before we discuss clipping lines, let's look at the simpler problem of clipping individual points.

If the x coordinate boundaries of the clipping rectangle are X_{min} and X_{max}, and the y coordinate boundaries are Y_{min} and Y_{max}, then the following inequalities must be satisfied for a point at (X, Y) to be inside the clipping rectangle:

$$X_{min} < X < X_{max}$$

and

$$Y_{min} < Y < Y_{max}$$

If any of the four inequalities does not hold, the point is outside the clipping rectangle.

Trivial Accept - save a line with both endpoints inside all clipping boundaries.
Trivial Reject - discard a line with both endpoints outside the clipping boundaries.
For all other lines - compute intersections of line with clipping boundaries.

Parametric representation of a line:

x = x1 + u (x2 - x1)
y = y1 + u (y2 - y1), and 0 <= u <= 1.

If the value of u for an intersection with a clipping edge is outside the range 0 to 1, then the line does not enter the interior of the window at that boundary. If the value of u is within this range, then the line does enter the interior of the window at that boundary.

14.2 Solve Simultaneous Equations

To clip a line, we need to consider only its endpoints, not its infinitely many interior points. If both endpoints of a line lie inside the clip rectangle (eg AB, refer to the first example), the entire line lies inside the clip rectangle and can be trivially accepted. If one endpoint lies inside and one outside(eg CD), the line intersects the clip rectangle and we must compute the intersection point. If both endpoints are outside the clip rectaangle, the line may or may not intersect with the clip rectangle (EF, GH, and IJ), and we need to perform further calculations to determine whether there are any intersections.

The brute-force approach to clipping a line that cannot be trivially accepted is to intersect that line with each of the four clip-rectangle edges to see whether any intersection points lie on those edges; if so, the line cuts the clip rectangle and is partially inside. For each line and clip-rectangle edge, we therefore take the two mathematically infinite lines that contain them and intersect them. Next, we test whether this intersection point is "interior" -- that is, whether it lies within both the clip rectangle edge and the line; if so, there is an intersection with the clip rectangle. In the first example, intersection points G' and H' are interior, but I' and J' are not.

14.3 The Cohen-Sutherland Line-Clipping Algorithm

The more efficient Cohen-Sutherland Algorithm performs initial tests on a line to determine whether intersection calculations can be avoided.

a) Steps for Cohen-Sutherland algorithm

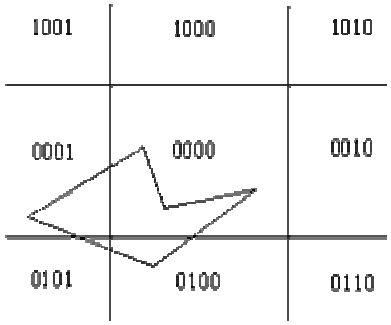
End-points pairs of the line are checked for trivial acceptance or trivial reject using outcode.

If not trivial-acceptance or trivial-reject, the line is divided into two segments at a clip edge.

Line is iteratively clipped by testing trivial-acceptance or trivial-rejected, and divided into two segments until completely inside or trivial-rejected.

b) Trivial acceptance/reject test

To perform trivial accept and reject tests, we extend the edges of the clip rectangle to divide the plane of the clip rectangle into nine regions. Each region is assigned a 4-bit code determined by where the region lies with respect to the outside halfplanes of the clip-rectangle edges. Each bit in the outcode is set to either 1 (true) or 0 (false); the 4 bits in the code correspond to the following conditions:



- Bit 1: outside halfplane of top edge, above top edge $Y > Y_{max}$
- Bit 2: outside halfplane of bottom edge, below bottom edge $Y < Y_{min}$
- Bit 3: outside halfplane of right edge, to the right of right edge $X > X_{max}$
- Bit 4: outside halfplane of left edge, to the left of left edge $X < X_{min}$

Conclusion

In summary, the Cohen-Sutherland algorithm is efficient when out-code testing can be done cheaply (for example, by doing bit-wise operations in assembly language) and trivial acceptance or rejection is applicable to the majority of line segments. (For example, large windows - everything is inside, or small windows - everything is outside).

14.4 Liang-Barsky Algorithm

Faster line clippers have been developed that are based on analysis of the parametric equation of a line segment, which we can write in the form:

$$x = x_1 + u\,\Delta x$$
$$y = y_1 + u\,\Delta y, \text{where } 0 <= u <= 1$$

Where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$. Using these parametric equations, Cryus and Beck developed an algorithm that is generally more efficient than the Cohen-Sutherland algorithm. Later, Liang and Barsky independently devised an even faster parametric line-clipping algorithm. Following the Liang-Barsky approach, we first write the point-clipping in a parametric way:

$$x_{min} <= x_1 + u\,\Delta x <= x_{max}$$
$$y_{min} <= y_1 + u\,\Delta y <= y_{max}$$

of these four inequalities can be expressed as

$$u * p_k <= q_k, \text{ for } k = 1, 2, 3, 4$$

Where parameters p and q are defined as:

$p_1 = -\Delta x, \quad q_1 = x_1 - x_{min}$
$p_2 = -\Delta x, \quad q_2 = x_{max} - x_1$
$p_3 = -\Delta y, \quad q_3 = y_1 - y_{min}$
$p_4 = -\Delta y, \quad q_4 = y_{max} - y_1$

Any line that is parallel to one of the clipping boundaries has $p_k = 0$ for the value of k corresponding to that boundary (k = 1, 2, 3, 4 correspond to the left, bottom, and top boundaries, respectively). If, for that value of k, we also find $q_k >= 0$, the line is inside the parallel clipping boundary.

When $p_k < 0$, the infinite extension of the line proceeds from the outside to the inside of the infitite extension of the particular clipping boundary. If $p_k > 0$, the line proceeds from the inside to the outside. For a nonzero value of $p_k = 0$, we can calculate the value of u that corresponds to the point where the infinitely extended line intersects the extension of boundary k as:

$$u = q_k / p_k$$

For each line, we can calculate values for parameters u_1 and u_2 that defines that part of the line that lies within the clip rectangle. The value of u_1 is determined by looking at the rectangle edges for which the line proceeds from the outer side to the inner side. ($p < 0$). For these edges we calculate $r_k = q_k / p_k$.

The value of u_1 is taken as the largest of the set consisting of o and the various values of r. Conversely, the value of u_2 is determined by examining the boundaries for which the line proceeds from inside to outside ($p > o$). A value of r_k is calculated for each of these boundaries and the value of u_2 is the minimum of the set consisting of 1 and the calculated r values. If $u_1 > u_2$, the line is completely outside the clip window and it can be rejected. Otherwise, the end points of the clipped line are calculated from the two values of parameter u.

This algorithm is presented in the following procedure. Line intersection parameters are initialized to the values $u_1 = 0$ and $u_2 = 1$. For each clipping boundary, the appropriate values for p and q are calculated and used by the function clipTest to determine whether the line can be rejected or whether the intersection parameters are to be adjusted.

When $p < 0$, the parameter r is used to update u_1; when $p < 0$, the parameter r is used to update u_2.

If updating u_1 or u_2 results in $u_1 > u_2$, we reject the line.

Otherwise, we update the appropriate u parameter only if the new value results in a shortening of the line.

When p = 0 and q < 0, we can discard the line since it is parallel to and outside of this boundary.

If the line has not been rejected after all four values of p and q have been tested, the endpoints of the clipped line are determined from values of u_1 and u_2.

Conclusion

In general, the Liang-Barsky algorithm is more efficient than the Cohen Sutherland algorithm, since intersection calculations are reduced. Each update of parameters u_1 and u_2 requires only one division; and window intersections of the line are computed only once, when the final values of u_1 and u_2 have computed. In contrast, the Cohen-Sutherland algorithm can repeatedly calculate intersections along a line path, even though the line may be completely outside the clip window, and, each intersection calculation requires both a division and a multiplication. Both the Cohen Sutherland and the Liang Barsky algorithms can be extended to three-dimensional clipping.