Computer Graphics

Lecture 16

Clipping Types Continued

Line Clipping

Figure 6-7 illustrates possible relationships between line positions and a standard rectangular clipping region. A line clipping procedure involves several parts. First, we can test a given line segment to determine whether it lies completely inside the clipping window. If it does not, we try to determine whether it lies completely outside the window. Finally, if we cannot identify a line as completely inside or completely outside, we must perform intersection calculations with one or more clipping boundaries. We process lines through the "inside-outside'' tests by checking the line endpoints. A line with both endpoints inside all clipping boundaries, such as the line from $\mathbf{P_1}$ to $\mathbf{P_2}$ is saved. A line with both endpoints outside any one of the clip boundaries (line P_3P_4 in Fig. 6-7) is outside the window. All other lines cross one or more clipping boundaries, and may require calculation of multiple intersection points. To minimize calculations, we try to devise clipping algorithms that can efficiently identify outside lines and reduce intersection calculations.

For a line segment with endpoints (x_1, y_1) and (x_2, y_2) and one or both endpoints outside the clipping rectangle, the parametric representation

$$\begin{aligned} x &= x_1 + u(x_2 - x_1) \\ y &= y_1 + u(y_2 - y_1), \qquad 0 \le u \le 1 \end{aligned} \tag{6-6}$$

could be used to determine values of parameter *u* for intersections with the clipping boundary coordinates. If the value of *u* for an intersection with a rectangle boundary edge is outside the range 0 to 1, the line does not enter the interior of the window at that boundary. If the value of *u* is within the range from 0 to 1, the line segment does indeed cross into the clipping area. This method can be applied to each clipping boundary edge in turn to determine whether any part of the line segment is to be displayed. Line segments that are parallel to window edges can be handled as special cases.

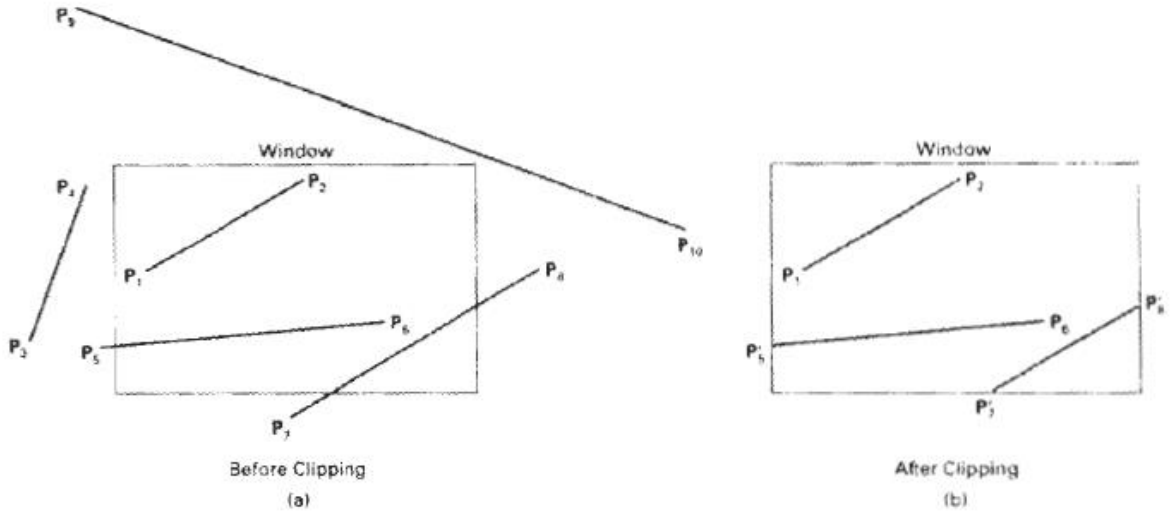

Figure 6-7
Line clipping against a rectangular clip window.

Cohen-Sutherland Line Clipping

This is one of the oldest and most popular line-clipping procedures. Generally, the method speeds up the processing of line segments by performing initial tests that reduce the number of intersections that must be calculated. Every line endpoint in a picture is assigned a four-digit binary code, called a region code that identifies the location of the point relative to the boundaries of the clipping rectangle. Regions are set up in reference to the boundaries as shown in Fig. 6-8. Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window: to the left, right, top, or bottom. By numbering the bit positions in the region code as 1 through 4 from right to left, the coordinate regions can be correlated with the bit positions as

bit 1: left
bit 2: right
bit 3: below
bit 4: above

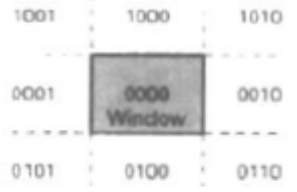

Figure 6-8
Binary region codes assigned to line endpoints according to relative position with respect to the clipping rectangle.

A value of 1 in any bit position indicates that the point is in that relative position; otherwise, the bit position is set to 0. If a point is within the clipping rectangle, the region code is 0000. A point that is below and to the left of the rectangle has a region code of 0101.

Bit values in the region code are determined by comparing endpoint coordinate values (x, y) to the clip boundaries. Bit 1 is set to 1 if $x < xw_{min}$ the other three bit values can be determined using similar comparisons. For languages in which bit manipulation is possible, region-code bit values can be determined with the following two steps: (1) Calculate differences between endpoint coordinates and clipping boundaries. (2) Use the resultant sign bit of each difference calculation to set the corresponding value in the region code. Bit 1 is the sign bit of $x - xw_{min}$; bit 2 is the sign bit of $xw_{max} - x$; bit 3 is the sign bit of $y - yw_{min}$; and bit 4 is the sign bit of $yw_{max} - y$.

Once we have established region codes for all line endpoints, we can quickly determine which lines are completely inside the clip window and which are clearly outside. Any lines that are completely contained within the window boundaries have a region code of 0000 for both endpoints, and we trivially accept these lines. Any lines that have a 1 in the same bit position in the region codes for each endpoint are completely outside the clipping rectangle, and we trivially reject these lines. We would discard the line that has a region code of 1001 for one endpoint and a code of 0101 for the other endpoint. Both endpoints of this line are left of the clipping rectangle, as indicated by the 1 in the first bit position of each region code. A method that can be used to test lines for total clipping is to perform the logical and operation with both region codes. If the result is not 0000, the line is completely outside the clipping region.

Lines that cannot be identified as completely inside or completely outside a clip window by these tests are checked for intersection with the window boundaries. As shown in Fig. 6-9, such lines may or may not cross into the window interior. We begin the clipping process for a line by comparing an outside endpoint to a clipping boundary to determine how much of the line can be discarded. Then the remaining part of the Line is checked against the other boundaries, and we continue until either the line is totally discarded or a section is found inside the window. We set up our algorithm to check line endpoints against clipping boundaries in the order left, right, bottom, top.
To illustrate the specific steps in clipping lines against rectangular boundaries using the Cohen-Sutherland algorithm, we show how the lines in Fig. 6-9 could be processed. Starting with the bottom endpoint of the line from P_1 to P_2

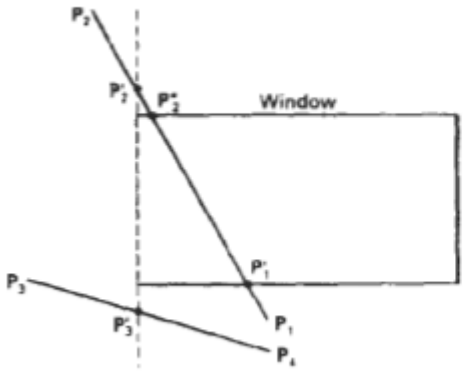

Figure 6-9
Lines extending from one coordinate region to another may pass through the clip window, or they may intersect clipping boundaries without entering the window.

We check $\mathbf{P_1}$ against the left, right, and bottom boundaries in turn and find that this point is below the clipping rectangle. We then find the intersection point $\mathbf{P_1'}$ with the bottom boundary and discard the line section from $\mathbf{P_1}$ to $\mathbf{P_1'}$. The line now has been reduced to the section from $\mathbf{P_1'}$ to $\mathbf{P_2}$ .Since $\mathbf{P_2}$ is outside the clip window, we check this endpoint against the boundaries and find that it is to the left of the window. Intersection point $\mathbf{P_2'}$ is calculated, but this point is above the window. So the final intersection calculation yields $\mathbf{P_2''}$, and the line from $\mathbf{P_1'}$ to $\mathbf{P_2''}$ is saved. This completes processing for this line, so we save this part and go on to the next line. Point $\mathbf{P_3}$ in the next line is to the left of the clipping rectangle, so we determine the intersection $\mathbf{P_3'}$, and eliminate the line section from $\mathbf{P_3}$ to $\mathbf{P_3'}$**.** By checking region codes for the line section from

P_3' to **P_4**, we find that the remainder of the line is below the clip window and can be discarded also.
Intersection points with a clipping boundary can be calculated using the **slope-intercept** form of the line equation. For a line with endpoint coordinates (x_1, y_1) and (x_2, y_2), the *y* coordinate of the intersection point with a vertical boundary can be obtained with the calculation

$$y = y_1 + m(x - x_1) \qquad (6\text{-}7)$$

where the x value is set either to xw_{min} or to xw_{max} and the slope of the line is calculated as $m = (y_2 - y_1) / (x_2 - x_1)$. Similarly, if we are looking for the intersection with a horizontal boundary, the x coordinate can be calculated as

$$x = x_1 + \frac{y - y_1}{m} \qquad (6\text{-}8)$$

with y set either to yw_{min},, or to yw_{max},.