Computer Graphics

Lecture 17

Polygon Clipping

To clip polygons, we need to modify the line-clipping procedures discussed in the previous section. A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments (Fig. 6-17), depending on the orientation of the polygon to the clipping window. What we really want to display is a bounded area after clipping, as in Fig. 6-18. For polygon clipping, we require an algorithm that will generate one or more closed areas that are then scan converted for the appropriate area fill. The output of a polygon clipper should be a sequence of vertices that defines the clipped polygon boundaries.

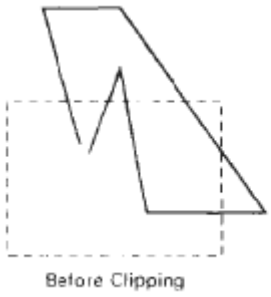

Figure 6-17
Display of a polygon processed by a line-clipping algorithm

Figure 6-18
Display of a correctly clipped polygon.

Sutherland-Hodgeman Polygon Clipping

We can correctly clip a polygon by processing the polygon bound jry as a whole against each window edge. This could be accomplished by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the initial set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices. The new set of vertices could then k successively passed to a right boundary clipper, a bottom boundary clipper, and a top

boundary clipper, as in Fig. 6-19. At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper.

There are four possible cases when processing vertices in sequence around the perimeter of a polygon. As each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests: (1) If the first vertex is outside the window boundary and the second vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list. (2) If both input vertices are inside the window boundary, only the second vertex is added to the output vertex list. (3) If the first vertex is inside the window boundary and the second vertex is outside, only the edge intersection with the window boundary is added to the output vertex list. (4) If both input vertices are outside the window boundary, nothing is added to the output list. These four cases are illustrated in Fig. 6-20 for successive pairs of polygon vertices. Once all vertices have been processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.

Figure 6-19
Clipping a polygon against successive window boundaries.

We illustrate this method by processing the area in Fig. 6-21 against the left window boundary. Vertices 1 and 2 are found to be on the outside of the boundary. Moving along to vertex 3, which is inside, we calculate the intersection and save both the intersection point and vertex 3. Vertices 4 and 5 are determined to be inside, and they also are saved. The sixth and final vertex is outside, so we find and save the intersection point. Using the five saved points, we would repeat the process for the next window boundary.

Implementing the algorithm as we have just described requires setting up storage for an output list of vertices as a polygon is clipped against each window boundary. We can eliminate the intermediate output vertex lists by simply clipping individual vertices at each step and passing the clipped vertices on to the next boundary clipper. This can be done with parallel processors or a single processor and a pipeline of clipping routines. A point (either an input vertex or a calculated intersection point) is added to the output vertex list only after it has been been determined to be inside or on a window boundary by all four boundary clippers. Otherwise, the point does not continue in the pipeline. Figure 6-22 shows a polygon and its intersection points with a clip window. In Fig. 6-23, we illustrate the progression of the polygon vertices in Fig. 6-22 through a pipeline of boundary clippers.

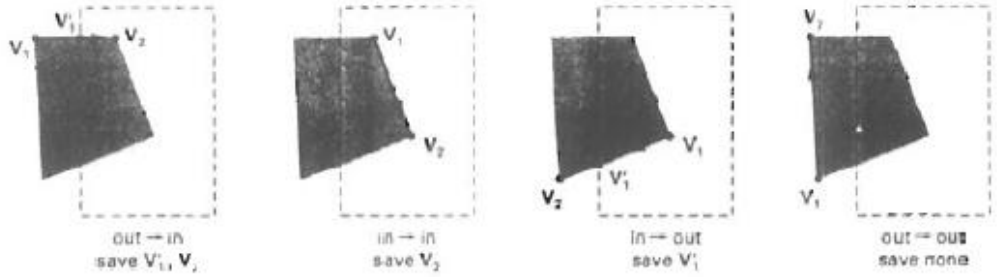

Figure 6-20
Successive processing of pairs of polygon vertices against the left window boundary.

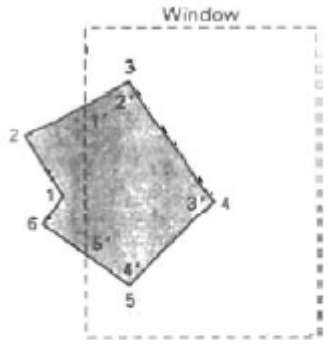

Figure 6-21
Clipping a polygon against the left boundary of a window, starting with vertex 1. Primed numbers are used to label the points in the output vertex list for this window boundary.

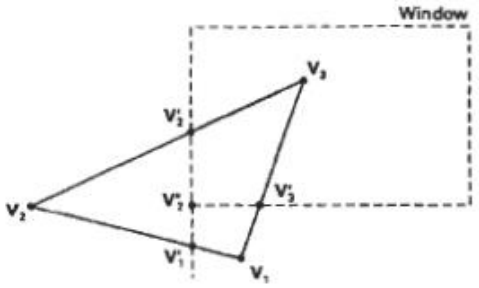

Figure 6-22
A polygon overlapping a rectangular clip window.

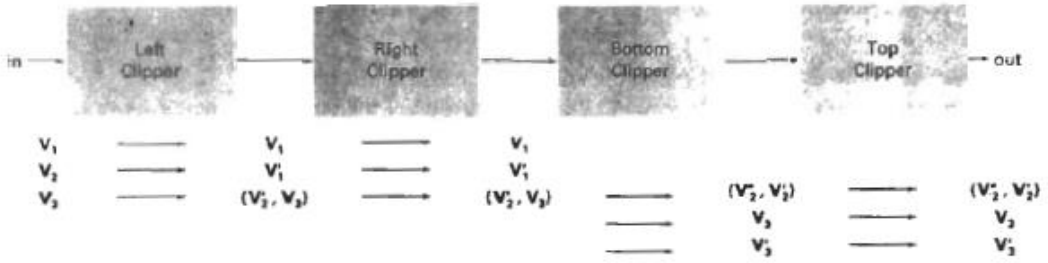

Figure 6-23
Processing the vertices of the polygon in Fig. 6-22 through a boundary-clipping pipeline. After all vertices are processed through the pipeline, the vertex list for the clipped polygon is (V_2'', V_2', V_3, V_3').

Convex polygons are correctly clipped by the Sutherland-Hodgeman algorithm, but concave polygons may be displayed with extraneous lines, as demonstrated in Fig. 6-24. This occurs when the clipped polygon should have two or more separate sections. But since there is only one output vertex list, the last vertex in the list is always joined to the first vertex. There are several things we could do to correctly display concave polygons. For one, we could split the concave polygon into two or more convex polygons and process each convex polygon separately. Another possibility is to modify the Sutherland-Hodgeman approach to check the final vertex list for multiple vertex points along any clip window boundary and correctly join pairs of vertices.

Curve Clipping

Areas with curved boundaries can be clipped with methods similar to those discussed in the previous .sections. Curve-clipping procedures will involve nonlinear equations, however, and this requires more processing than for objects with linear boundaries.

The bounding rectangle for a circle or other curved object can be used first to test for overlap with a rectangular clip window. If the bounding rectangle for the object is completely inside the window, we save the object. If the rectangle is determined to be completely outside the window,

we discard the object. In either case, there is no further computation necessary. But if the bounding rectangle test fails, we can look for other computation-saving approaches. For a circle, we can use the coordinate extents of individual quadrants and then octants for preliminary testing before calculating curve-window intersections. For an ellipse, we can test the coordinate extents of individual quadrants. Figure 6-27 illustrates circle clipping against a rectangular window.

Similar procedures can be applied when clipping a curved object against a general polygon clip region. On the first pass, we can clip the bounding rectangle of the object against the bounding rectangle of the clip region. If the two regions overlap, we will need to solve the simultaneous line-curve equations to obtain the clipping intersection points.

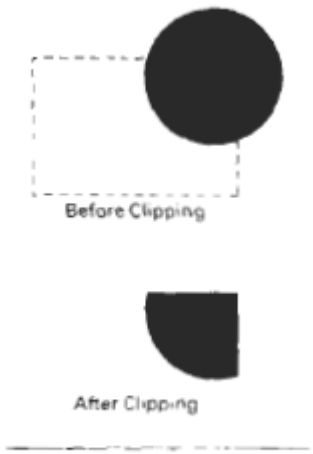

Figure 6-27
Clipping a filled circle.

Text Clipping

There are several techniques that can be used to provide text clipping in a graphics package. The clipping technique used will depend on the methods used to generate characters and the requirements of a particular application.

The simplest method for processing character strings relative to a window boundary is to use the all-or-none string-clipping strategy shown in Fig. 6-28. If all of the string is inside a clip window, we keep it. Otherwise, the string is discarded. This procedure is implemented by considering a bounding rectangle around the text pattern. The boundary positions of the rectangle are then compared to the window boundaries, and the string is

rejected if there is any overlap. This method produces the fastest text clipping.

An alternative to rejecting an entire character string that overlaps a window boundary is to use the all-or-none character-clipping strategy. Here we discard only those characters that are not completely inside the window (Fig. 6-29). In this case, the boundary limits of individual characters are compared to the window. Any character that either overlaps or is outside a window boundary is clipped.

A final method for handling text clipping is to clip the components of individual characters. We now treat characters in much the same way that we treated lines. If an individual character overlaps a clip window boundary, we clip off the parts of the character that are outside the window (Fig. 6-30). Outline character fonts formed with line segments can be processed in this way using a line-clipping algorithm. Characters defined with bit maps would be clipped by comparing the relative position of the individual pixels in the character grid patterns to the clipping boundaries.

Figure 6-28
Text clipping using a bounding rectangle about the entire string.