

Clipping

1 Purpose

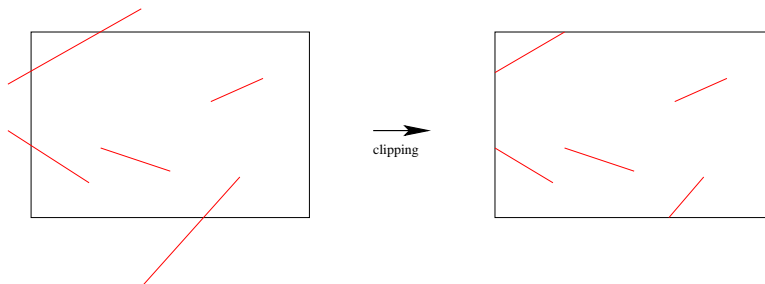


Figure 1: The purpose of clipping.

Plotting pixels and computing pixels to be plotted takes time. Especially in large scenes with lots of small primitives, many of them project partially or completely outside the viewport or are outside or stick out of the viewing volume. Such primitives or their parts which stick out need not be drawn.

Clipping allows to compute the (parts of) primitives which are inside the viewing volume or viewport.

2 Four stages

An approach which is particularly attractive for hardware implementation (since it is of pipeline type), decomposes the process of clipping against a rectangular viewport into four stages, each one being clipping against a half-plane (Figure 2). We'll assume that the bounding lines of the half-planes are parallel to the coordinate axes.



Figure 2: Clipping pipeline.

3 One stage: Clipping against a half-plane

3.1 Lines

First, decide whether the line intersects the line l bounding the half-plane or not. If it does not and the line is outside, it's clipped to nothing. If it is inside then no clipping is necessary. This test is very cheap computationally (if l is parallel to the coordinate axes, it can be done by comparing the coordinates; see Figure 3 for an example).

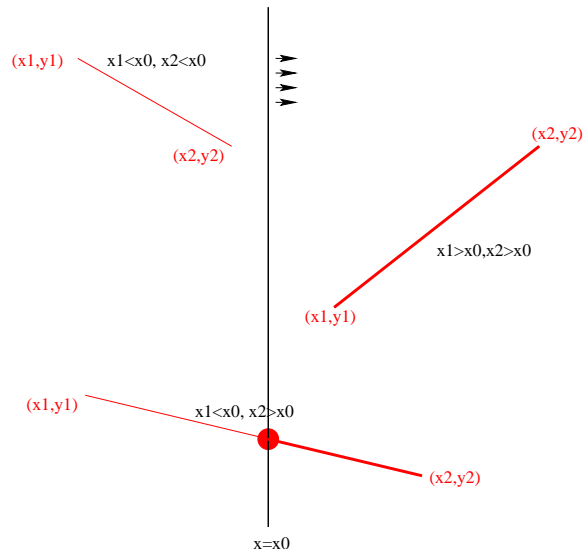


Figure 3: Clipping a line against a halfplane.

If the line intersects l , we need to compute the intersection point to determine its clipped part.

3.2 Polygons

Note that clipping a polygon may potentially change its number of edges. To list the vertices of a polygon clipped against a half-plane we need to do the following. Walk around the boundary of the polygon, starting from some vertex until you return back to that very vertex. Whenever you encounter a vertex which is inside the half-plane, output it. Do not output vertices which are outside the half-plane. Also, whenever the line bounding the half-plane is crossed, output the crossing point. This should be implemented in such a way that the intersection point is computed only when necessary (apply a comparison test as in the case of lines to see whether an edge intersects the boundary or not before you start computing anything). See Figure 4 for an example. Note that this procedure works in the case of non-convex polygons as well.

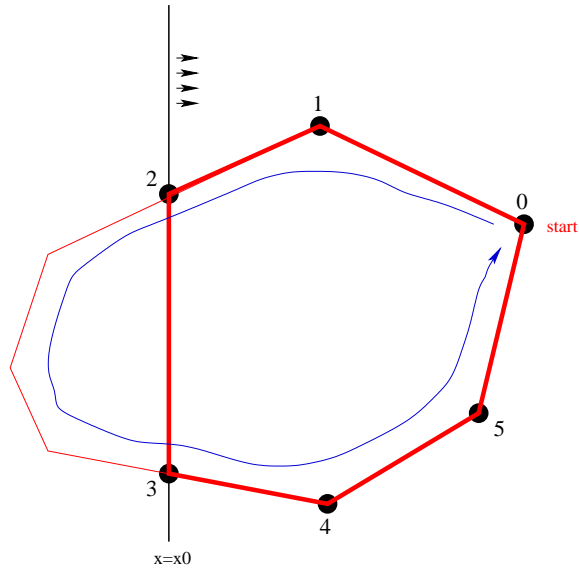


Figure 4: Clipping a polygon against a halfplane. Thick red lines show the clipped polygon; the black dots are the output vertices and crossings i.e. the vertices of the clipped polygon, with the numbers indicating their order.

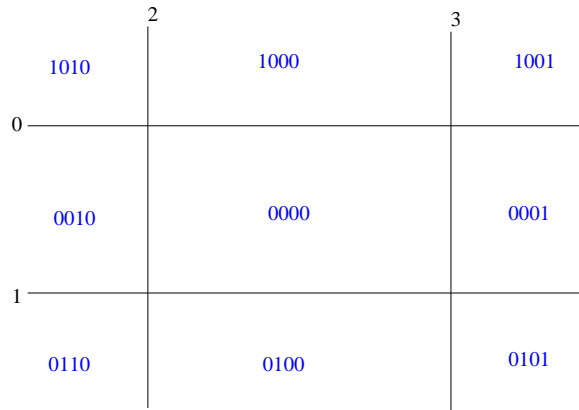


Figure 5: Outcodes on different regions of the plane.

4 Optimizations

There are ways to quickly decide that no clipping is necessary. One of them is based on the *outcode*. Let's say that we are clipping against a rectangular viewport, bounded by 4 lines and let's give the lines labels 0...3 as in Figure 5, left. An outcode of a point in the plane is defined as a four-bit code, the i -th bit indicating whether the point lies on the same side (then it's zero) of the i -th line

as the viewport or not (1). Clearly, outcodes for all points in the same region into which the lines through the edges of the rectangle split the plane are the same; they are shown in Figure 5. Now, it's not hard to see that if the outcodes of both endpoints of a line segment (or all three vertices of a triangle) are zero then that line (triangle) is completely inside the viewport. Thus, a clipping routine can just pass it on without any further calculations. If the bitwise AND of outcodes of both endpoints of a line segment (or all the three vertices of a triangle) is nonzero, then that line (triangle) is completely outside the viewport, which means that it can be skipped.

5 3D case: clipping against the view volume

After the perspective projection, the viewing volume is rectilinear and all its bounding faces are parallel to the coordinate planes. The same procedure as in 2D can be applied to do the clipping (we have 6 stages instead of 4, since there are 6 planes bounding the view volume).

Clipping can be performed before the scan-conversion stage and after the perspective transformation.