Clipping

Used to identify which primitives (or which parts of a primitive) lie inside the view volume, and therefore need to be projected onto the view plane.

Start with the simple 2D case, clipping polygons to a rectangular viewport running from \((x_{min}, y_{min})\) to \((x_{max}, y_{max})\).

One standard technique is to repeatedly clip the line segments that make up the boundary of the polygon, rather than clipping the polygon directly.

Brute Force Method

To clip a line, we need only consider its endpoints. If both endpoints lie within the viewport, the line can be trivially accepted. If one endpoint lies inside and one lies outside, the line intersects the viewport and we must compute the intersection point. If both endpoints lie outside, the line may (or may not) intersect the viewport, and further testing is needed.
In the brute-force approach, for any line we cannot trivially accept, we intersect it against the four edges of the viewport, then check to see if the intersection is "interior" to both line segment and viewport edge. If so, the line segment intersects the viewport at that position.

This approach requires solving two simultaneous equations for each viewport edge. We use parametric representations to denote the fixed-length line segment and viewport edge:

\[ x = x_0 + t(x_1 - x_0) \Rightarrow 0 \leq t \leq 1 \]
\[ y = y_0 + t(y_1 - y_0) \]

Two sets of equations are solved for time and tedge. If \(0 \leq t \leq 1\), line segment intersects viewport.
Must also pre-check for line segment parallel to viewport edge.

In summary, very expensive due to numerous intersection tests. Must be done for all polys in scene each time we render, so expense multiplies quickly.

Cohen-Sutherland Line Clipping Algorithm

In Cohen-Sutherland, if line segment cannot be trivially accepted, region checks are done to see if segment can be trivially rejected. If both endpoints lie on the same (outer) side of a half-plane formed by viewport edge, line cannot intersect viewport and can be immediately rejected.

E.g., segment EF, both E and F lie to the left of the left edge of the viewport, so segment can be immediately clipped. Also works if segment's endpoints lie above top edge, below bottom edge, or right of right edge.

If line segment cannot be trivially accepted or rejected, it is subdivided into 2 pieces, one of which CAN be trivially rejected. Remaining piece is then retested against viewport.
In order to perform its tests, the viewport edges are extended to divide the viewplane into 9 regions. Each region is assigned a bit code based on its position relative to the outside half-plane of each of the 4 edges:

- bit 1: 1 if outside top edge, $y > y_{max}$
- bit 2: 1 if outside bottom edge, $y < y_{min}$
- bit 3: 1 if outside right edge, $x > x_{max}$
- bit 4: 1 if outside left edge, $x < x_{min}$

For any line segment $AB$, assign bit codes $bca$ to endpoint $A$, $bcb$ to endpoint $B$.

- If $bca = bcb = 0$, trivially accept
- If $bca \& bcb \neq 0$, trivially reject
  (because both endpoints outside at least one common half-plane)

Otherwise, segment must be subdivided into 2 parts: one to be trivially rejected, one to be retested. To do this:
1. Select an endpoint that lies outside viewport (at least one such endpoint exists, otherwise segment would be trivially accepted).

2. Select a viewport edge where outside endpoint lies outside halfplane formed by that edge.

3. Intersect line segment with viewport edge.

4. Portion of segment from intersection to outside endpoint can now be trivially rejected (since both points are outside the halfplane formed by viewport edge it intersected against).

5. Remaining portion of line segment can now be rejected.

Ex., line segment CD:

\[
\text{Ex., line segment CD:}
\]

\[
\begin{array}{c}
\text{Ex., line segment CD:}
\end{array}
\]

Since \( b_{cc} \neq \emptyset \) and \((b_{cc} \& b_{cd}) = \emptyset\), cannot trivially accept or reject.
1. Choose D as endpoint outside viewport.
2. Intersect D against top edge.
3. Intersection point is D'.
4. Trivially reject D'.
5. Reject CD'.

Now bc_1 = 01010 & CD' is trivially accepted.

E.g. Line segment IJ:

\[ \begin{array}{c|c|c}
\text{I} & \text{J} & \text{bc}_1 = 01010 \\
\hline
1011 & 1010 & 1010 \\
\end{array} \]

Since bc_2, bc_3 \neq 0 and (bc_1 & bc_2) = 0, cannot trivially accept or reject.

Arbitrarily select I at the outside endpoint, clip it against bottom edge (since second bit of bc_3 is 1).
IJ is trivially rejected, J'I is retested.

Now bc_3 = 1010 and (bc_3 & bc_1) = 01010 \Rightarrow J'I is trivially rejected, bc_3 is entirely to right of right viewport edge.

Notice if we'd decided to start with J rather than I, we would have intersected against right.
edge, trivially rejected \( IT \), rejected \( IT \),
and trivially rejected it as lying below bottom edge of viewport.

Cohen-Sutherland algorithm is nice because it
deals efficiently with two common cases:

1. Large viewport relative to scene, so almost
   all line segments trivially accepted

2. Small viewport relative to scene (picking),
   so almost all line segments trivially rejected

Also tries to minimize number of intersection
tests needed to classify a line segment.