Introduction

It is a procedure which identifies that portion of a picture which is either inside or outside a region is referred to as a **clipping algorithm** or **clipping**.

The region against which an object is to be clipped is called clipping window.

Viewing in 2D

- In 2D, a ‘world’ consists of an infinite plane, defined in ‘world’ coordinates, i.e metres, Kilometers, etc.
- We need to pick an area of the 2D plane to view, referred to as the ‘*window*’.
- On our display device, need to allocate an area for display, referred to as the ‘*viewport’* in device specific coordinates.
  - Clip objects outside of window.
  - Translate to fit viewport.
  - Scale to device coordinates.

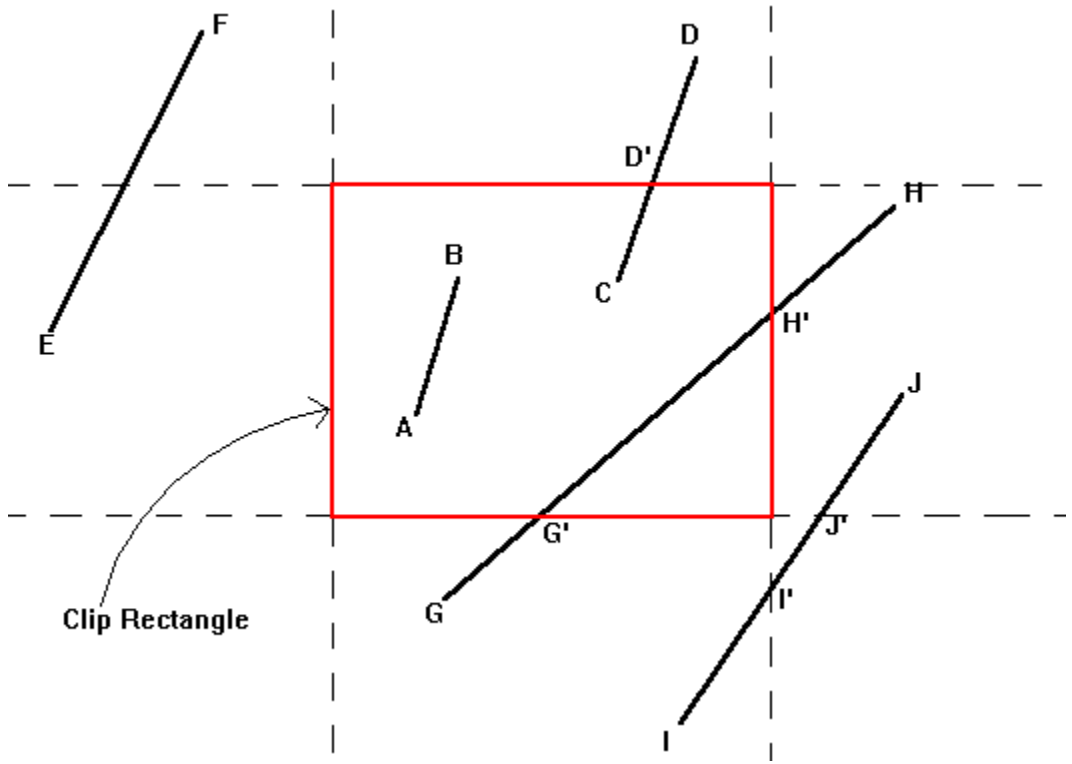
Viewing in 2D

250 M

45 M

Clipping in 2D.

- Need to clip primitives against the sides of the window.
  - e.g lines



Clipping Individual Points

If the x coordinate boundaries of the clipping rectangle are Xmin and Xmax, and the y coordinate boundaries are Ymin and Ymax, then the following inequalities must be satisfied for a point at (X,Y) to be inside the clipping rectangle:

$Xmin < X < Xmax$ and
$Ymin < Y < Ymax$

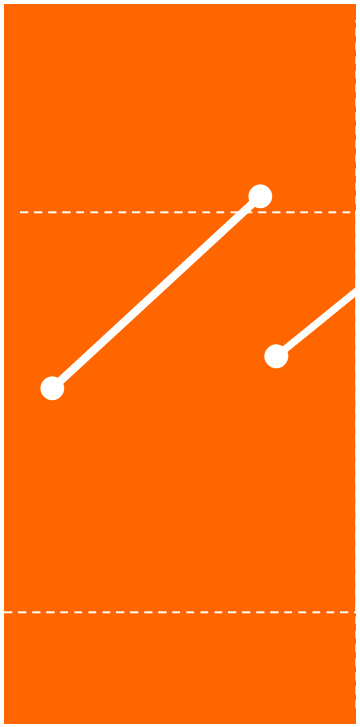
If any of the four inequalities does not hold, the point is outside the clipping rectangle.

Cohen-Sutherland Line Clipping

The Cohen-Sutherland line clipping algorithm detects and dispenses with two common and trivial cases.
To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is **trivially accepted** and needs no clipping. On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is **trivially rejected** and needs to be neither clipped nor displayed.

Trivial acceptance.

Trivial rejection.



Cohen-Sutherland algorithm

- Efficient method of accepting or rejecting lines that don't intersect the window edges.
- Assign a binary 4 bit code to each vertex :

- First bit set **1** : Point lies to **left** of window $x < x_{min}$
- Second bit set **1** : Point lies to **right** of window $x > x_{max}$
- Third bit set **1** : Point lies below(**bottom**) window $y < y_{min}$
- fourth bit set **1** : Point lies above(**top**) window $y > y_{max}$

The sequence for reading the codes' bits is **LRBT** (Left, Right, Bottom, Top).

– 4-bit code called: *Outcode*

Cohen-Sutherland 2D outcodes

Cohen-Sutherland algorithm

Cohen-Sutherland algorithm

Cohen-Sutherland algorithm.

- The line segment's endpoints are tested to see if the line can be **trivally accepted or rejected.**
- If the line cannot be trivally accepted or rejected, an intersection of the line with a window edge is determined and the trivial reject/accept test is repeated.
- This process is continued until the line is accepted.

1. Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$
2. Compute the 4-bit codes for each endpoint.

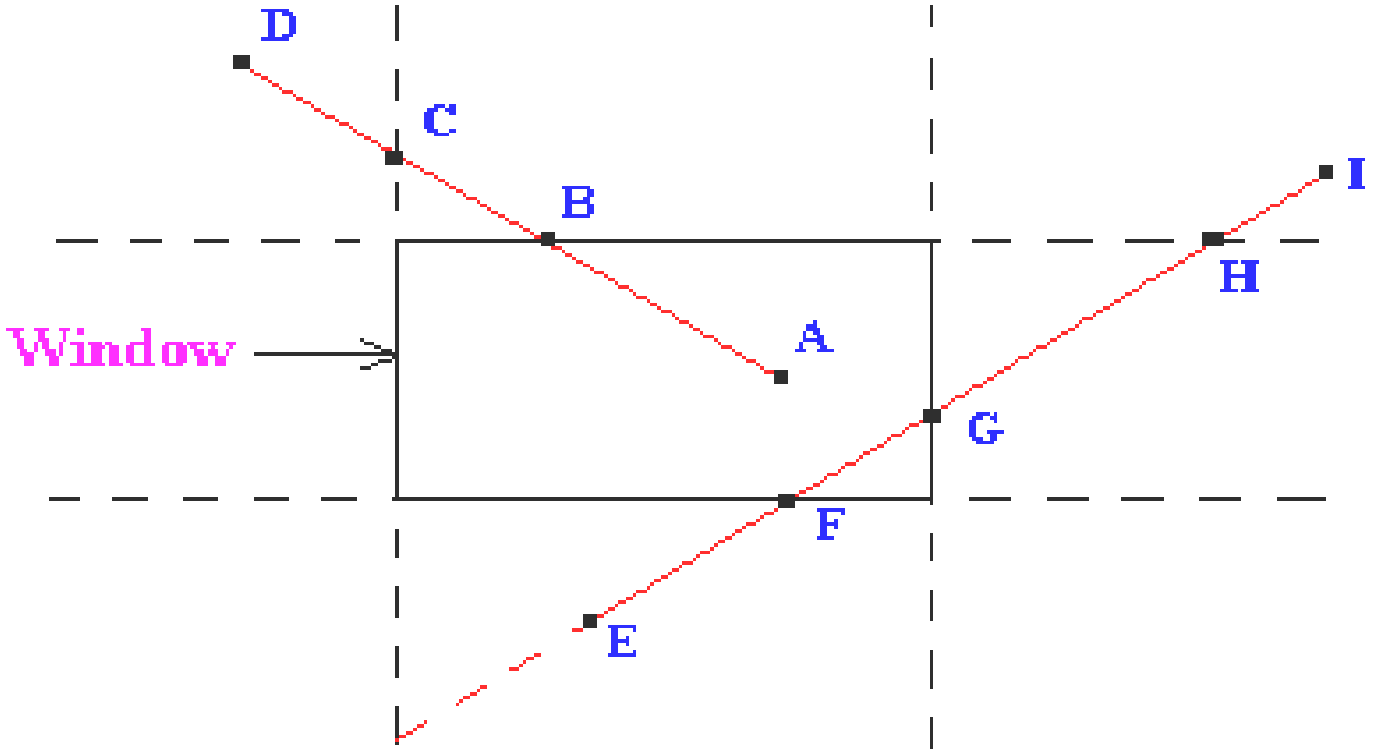
   If both codes are **0000** (bitwise OR of the codes yields 0000) line lies completely **inside** the window: pass the endpoints to the draw routine.

   If both codes have a 1 in the same bit position (bitwise AND of the codes is **not** 0000), the line lies **outside** the window. It can be trivially rejected.

3. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be **clipped** at the window edge before being passed to the drawing routine.
4. Examine one of the endpoints, say $P_1 = (x_1, y_1)$. Read P_1's 4-bit code in order: **Left**-to-**Right**, **Bottom**-to-**Top**.
5. When a set bit (1) is found, compute the **intersection** **I** of the corresponding window edge with the line from P_1 to P_2. Replace P_1 with **I** and repeat the algorithm.

Illustration of Line Clipping

Before Clipping



Consider the line segment **AD**

Point **A** has an outcode of **0000** and point **D** has an outcode of **1001**. The logical AND of these outcodes is zero; therefore, the line cannot be trivally rejected. Also, the logical OR of the outcodes is not zero; therefore, the line cannot be trivally accepted. The algorithm then chooses **D** as the outside point (its outcode contains 1's). By our testing order, we first use the top edge to clip **AD** at **B**. The algorithm then recomputes **B**'s outcode as **0000**. With the next iteration of the algorithm, **AB** is tested and is trivially accepted and displayed.

Consider the line segment **EI**

Point **E** has an outcode of **0100**, while point **I**'s outcode is **1010**. The results of the trivial tests show that the line can neither be trivally rejected or accepted. Point **E** is determined to be an outside point, so the algorithm clips the line against the bottom edge of the window. Now line **EI** has been clipped to be line **FI**. Line **FI** is tested and cannot be trivially accepted or rejected. Point **F** has an outcode of **0000**, so the algorithm chooses point **I** as an outside point since its outcode is**1010**. The line **FI** is clipped against the window's top edge, yielding a new line **FH**. Line **FH** cannot be trivally accepted or rejected. Since **H**'s outcode is **0010**, the next iteration of the algorthm clips against the window's right edge, yielding line **FG**. The next iteration of the algorithm tests **FG**, and it is trivially accepted and display.

Intersection of Line Segment

If the line cannot be trivially accepted or rejected, we must compute the intersection of the line with an appropriate window edge.

We need two equations for the line

- $y = mx + b$ where b,c are constant
- $x = 1/_m y + c$

Find the intersection with the top edge of the window:

$$y = T \text{ (Top)} \quad \text{or} \quad y = y_{max}$$
$$x = 1/_m T + c$$

Then, find the intersection with the bottom edge:

$$y = B \text{ (Bottom)} \quad \text{or} \quad y = y_{min}$$
$$x = 1/_m B + c$$

Now, find the intersection with the left edge:

$$x = L \text{ (Left)} \quad \text{or} \quad x = x_{min}$$
$$y = mL + b$$

Finally, find the intersection with the right edge:

$$x = R \text{ (Right)} \quad \text{or} \quad x = x_{max}$$
$$y = mR + b$$

There is no significance in the sequence of which edge to examine.

Application

In **3D graphics**, in a city street scene the computer may have model, texture, and shader data in memory for every building in the city; but since the camera viewing the scene only sees things within, say, a 90° angle, or field of view, the computer does not need to transform, texture, and shade the buildings that are behind the camera, nor those which are far enough to the side that they are off the screen. The clipping algorithm lets the rendering code skip all consideration of those buildings, and the program runs faster.

Scope of Research

The Nicholl-Lee-Nicholl algorithm is a fast line clipping algorithm that reduces the chances of clipping a single line segment multiple times, as may happen in the Cohen-Sutherland algorithm. The clipping window is divided into a number of different areas, depending on the position of the initial point of the line to be clipped.