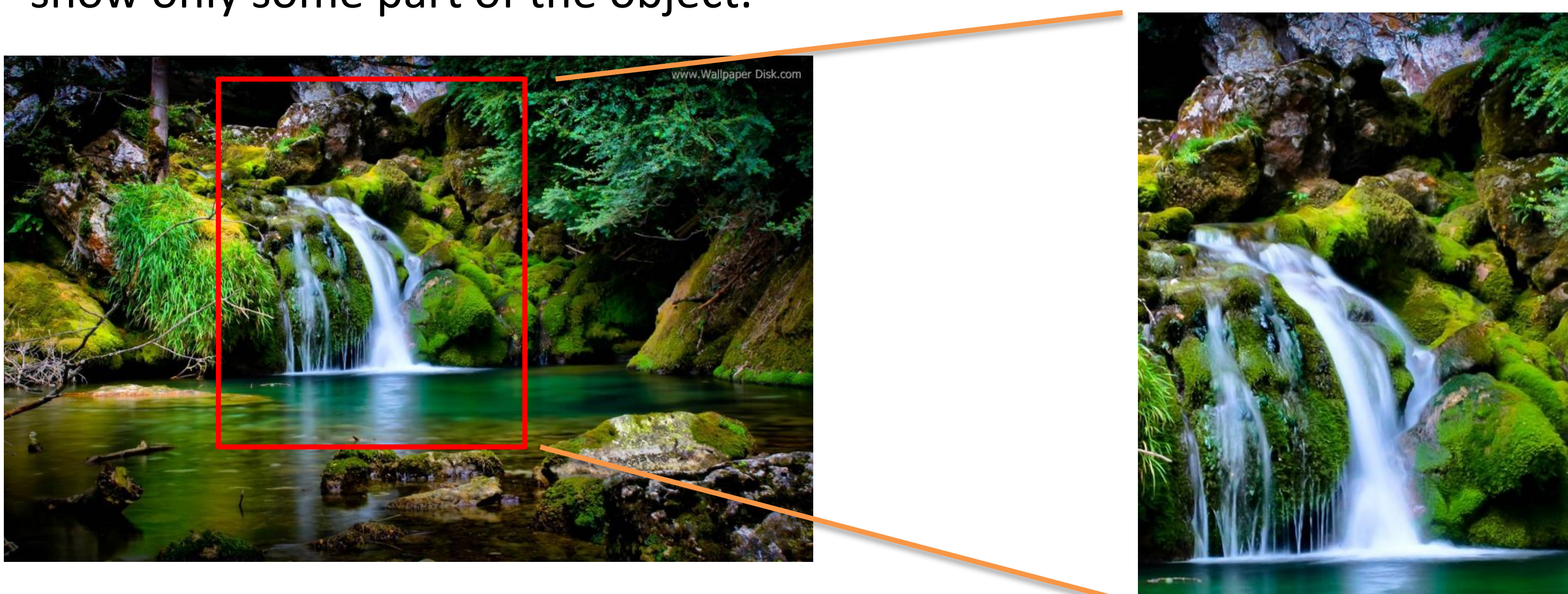
Computer Graphics

Lecture 10

Clipping Algorithms

- All objects in the real world have size. We use a unit of measure to describe both the size of an object as well as the location of the object in the real world.

- Sometimes the complete picture of object in the world coordinate system is too large and complicate to clearly show on the screen, and we need to show only some part of the object.



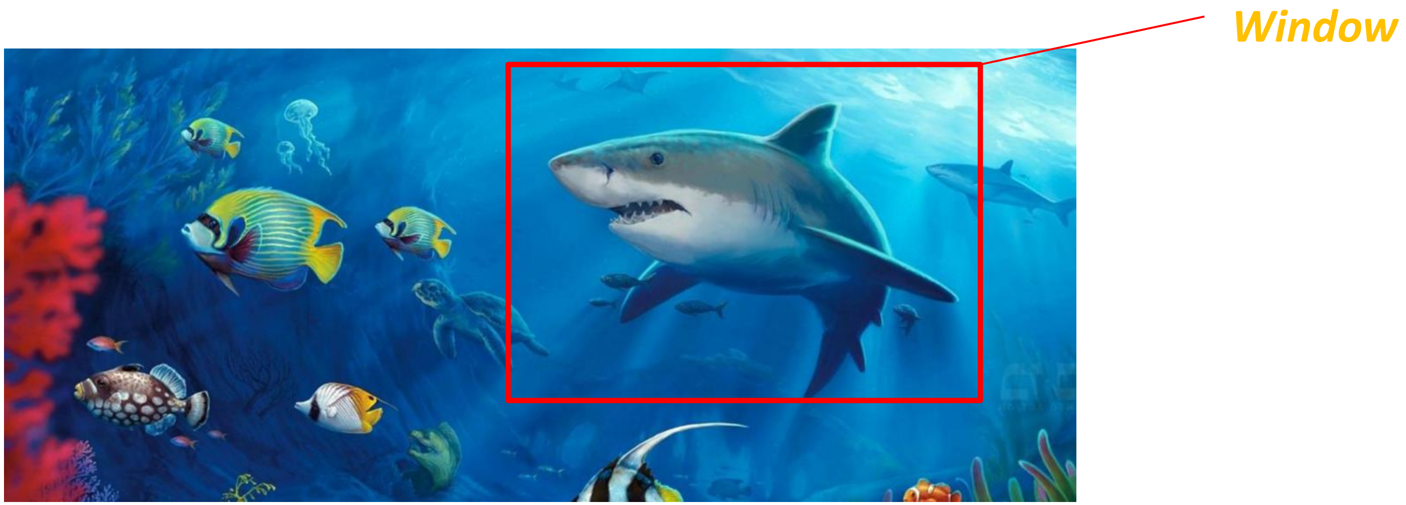
https://www.pinterest.com

Window

The capability that show some part of object internal a specify window is called **windowing**

Rectangular region in a world coordinate system is called **window**

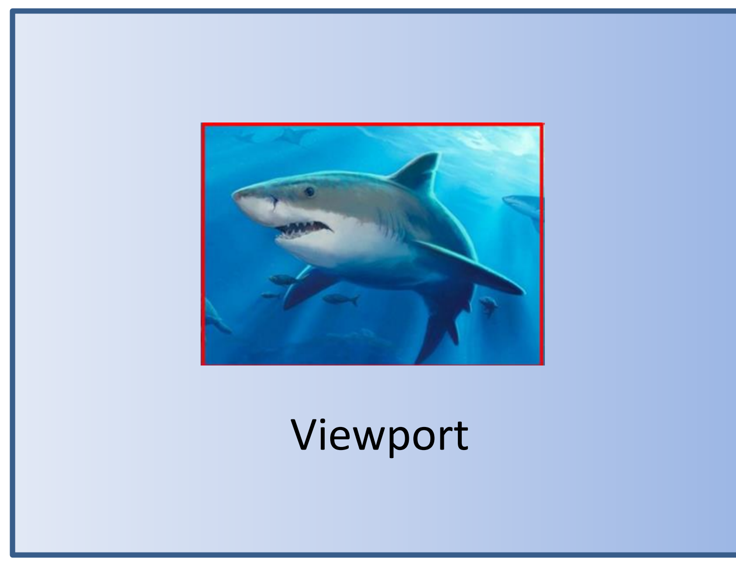
- This is the coordinate system used to locate an object in the natural world
- The world coordinate system does not depend on a display device



Viewport

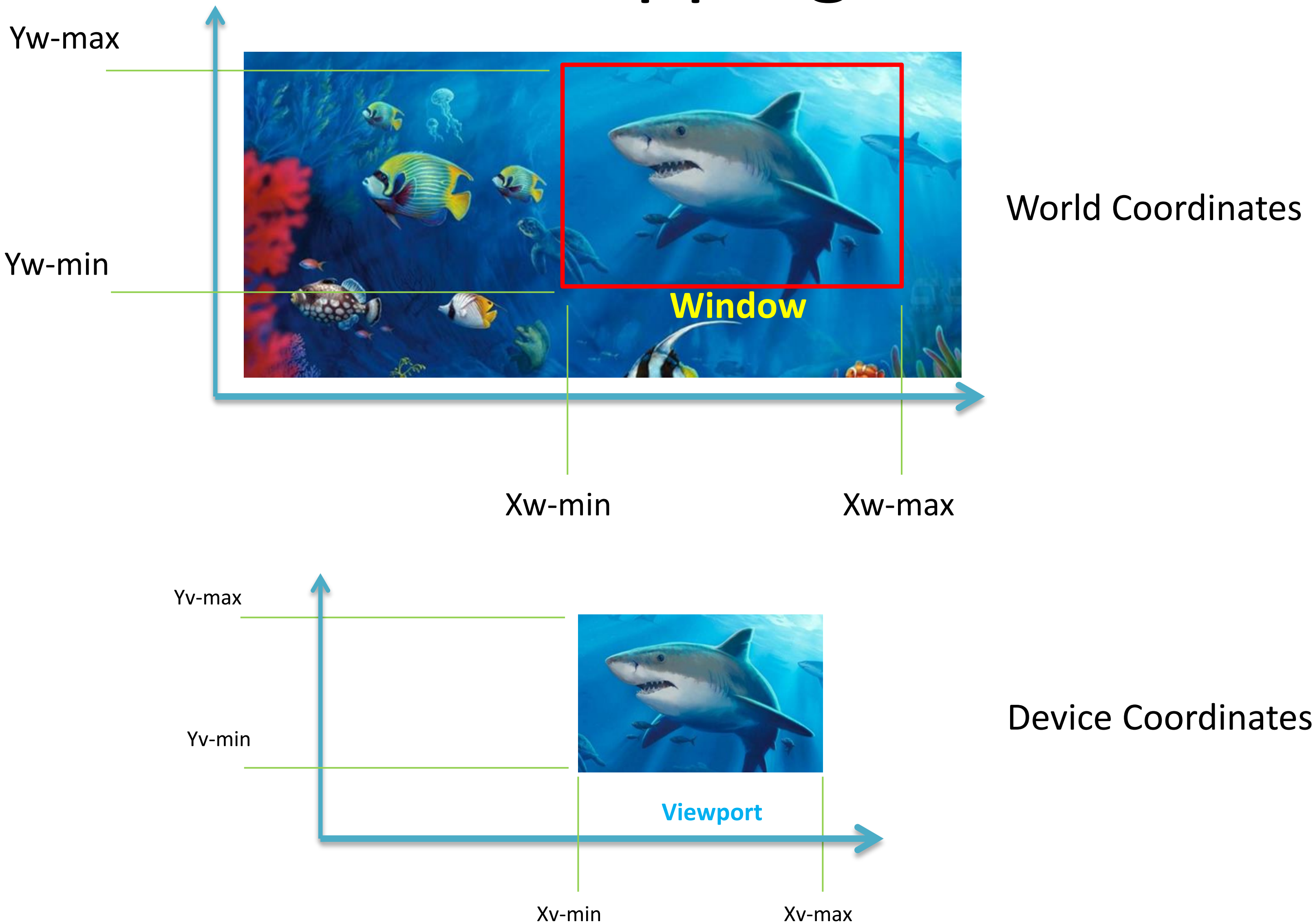
A **Viewport** is the section of the screen where the images covered by the window on the world coordinate system will be drawn.

The viewport uses the **screen coordinate system** so this transformation is from the world coordinate system to the screen coordinate system (Mapping).



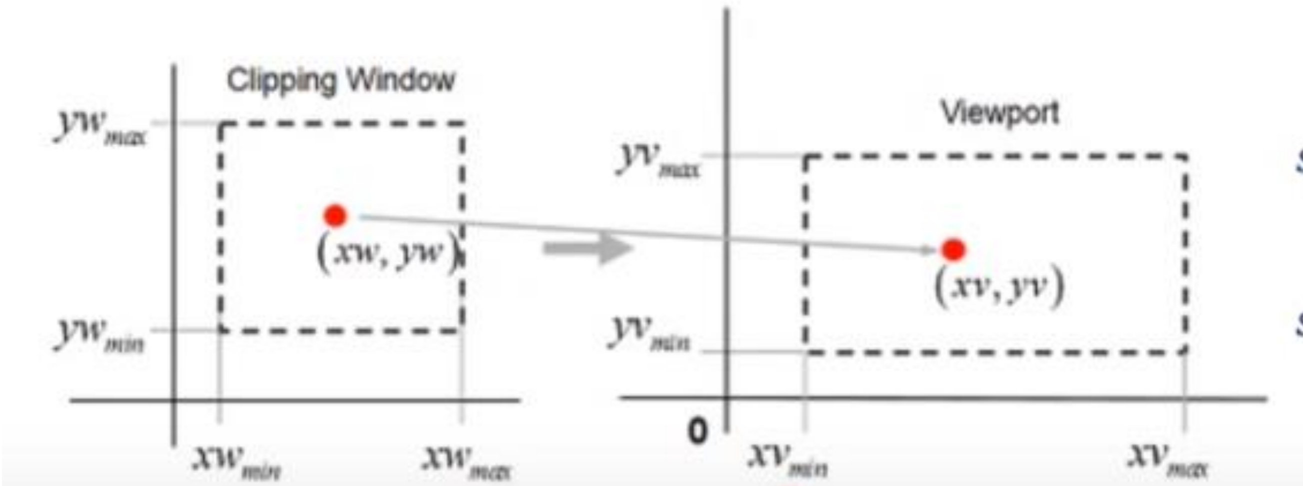
Screen

2D Mapping


Yw-max
Yw-min
Window
World Coordinates
Xw-min
Xw-max
Yv-max
Yv-min
Viewport
Device Coordinates
Xv-min
Xv-max

2D Mapping

Mapping can be done by scaling the window down to the viewport



Scaling factor *Sx*, *Sy*

$$s_x = \frac{x_v max - x_v min}{x_w max - x_w min} \rightarrow (1)$$

$$s_y = \frac{y_v max - y_v min}{y_w max - y_w min} \rightarrow (2)$$

2D Mapping

$$s_x = \frac{x_v max - x_v min}{x_w max - x_w min} \rightarrow (1)$$

$$s_y = \frac{y_v max - y_v min}{y_w max - y_w min} \rightarrow (2)$$

$$\frac{x_v - x_v min}{x_v max - x_v min} = \frac{x_w - x_w min}{x_w max - x_w min} \rightarrow (3)$$

$$\frac{y_v - y_v min}{y_v max - y_v min} = \frac{y_w - y_w min}{y_w max - y_w min} \rightarrow (4)$$

2D Mapping

$$x_v - x_v min = \frac{x_w - x_w min}{x_w max - x_w min} . x_v max - x_v min$$

Sub (1)

$$x_v - x_v min = (x_w - x_w \text{min}). \, s_x$$

$$x_v = x_v min + (x_w - x_w \text{min}). \, s_x \dashrightarrow (6)$$

Similarly

$$y_v = y_v min + (y_w - y_w \text{min}). \, s_y \dashrightarrow (7)$$

2D Mapping Example

Example

- Consider the window is located from 0 to 100 and a point is located in (30,30). Identify the point new location in the viewport. Consider the viewport size as 0 to 50.

Solution:

$x_w min=0$ $x_w max=100$ $y_w min=0$ $y_w max=100$

$x_v min=0$ $x_v max=50$ $y_v min=0$ $y_v max=50$

$P(x,y)=(30,30)$

Xv = ?
Yv = ?

Clipping

When a window is "placed" on the world, only certain objects and parts of objects can be seen. Points and lines which are outside the window are "cut off" from view.

This process of "cutting off" parts of the image of the world is called **Clipping**

In clipping, we examine each line to determine

- whether or not it is completely inside the window
- completely outside the window
- crosses a window boundary.

If inside the window, the line is displayed. If outside the window, the lines and points are not displayed.

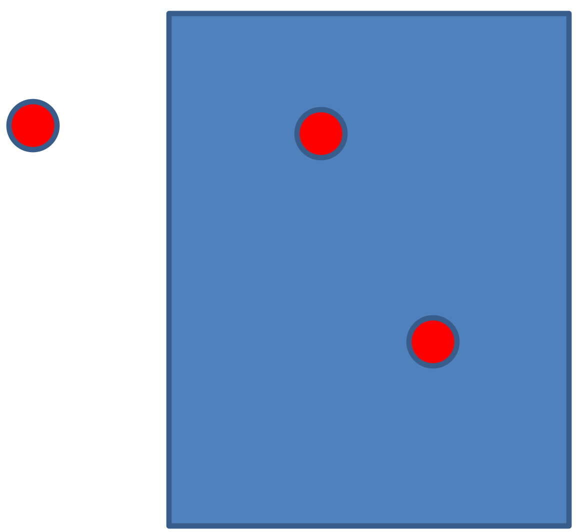
If a line crosses the boundary, we must determine the point of intersection and display only the part which lies inside the window.

Point clipping

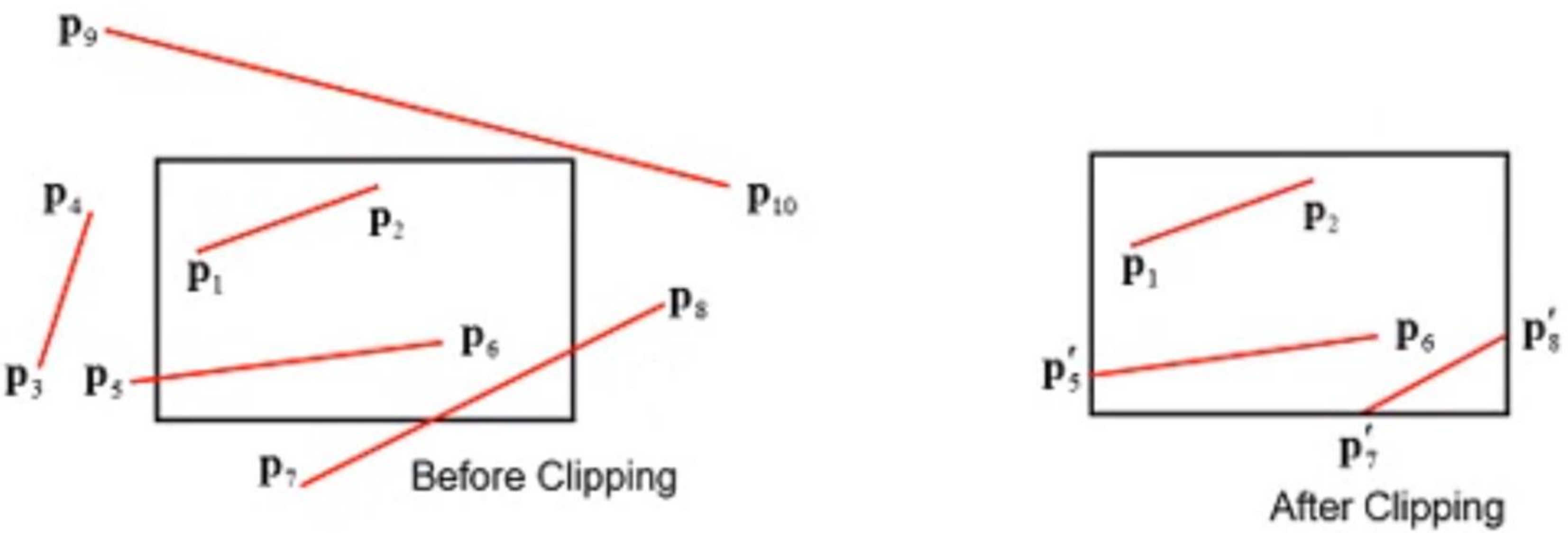
Assume a point P(x,y) does not satisfy the following conditions will be clipped away

$$x_w min \le x \le x_w max$$

$$y_w min \le y \le y_w max$$



Line Clipping



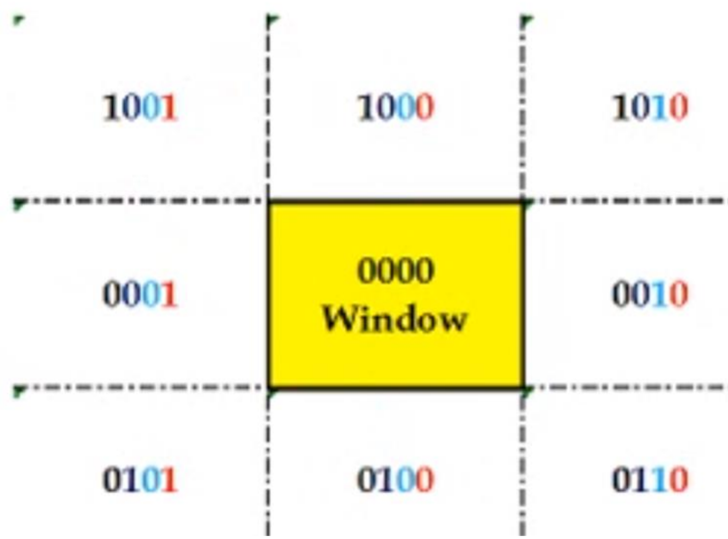
- Line P_3P_4, P_9P_{10} *completely outside Rejected*
- Line P_1P_2 *completely inside Accepted*
- Line P_5P_6, P_7P_8 partially inside and partially outside

Cohen-Sutherland Line Clipping

Each of the nine regions associated with the window is assigned a 4-bit code to identify the region. Each bit in the code is set to either a 1(true) or a 0(false). If the region is to the left of the window, the first bit of the code is set to 1. If the region is to the top of the window, the second bit of the code is set to 1. If to the right, the third bit is set, and if to the bottom, the fourth bit is set. The 4 bits in the code then identify each of the nine regions as shown below.

Every line End points is assigned a four digit Binary code Region code/out code/Area code

$b_1 = 1$ *if the point is in the left side of the window*
$b_2 = 1$ *if the point is in the Right side of the window*
$b_3 = 1$ *if the point is in the Bottom side of the window*
$b_4 = 1$ *if the point is in the Top side of the window*



Cohen-Sutherland Line Clipping

For any endpoint (x , y) of a line, the code can be determined that identifies which region the endpoint lies. The code's bits are set according to the following conditions:

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

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

Cohen-Sutherland Line Clipping

Once the codes for each endpoint of a line are determined, the logical **AND** operation of the codes determines if the line is completely outside of the window.

If the logical AND of the endpoint codes is **not zero**, the line can be trivially rejected.

Ex:

if an endpoint had a code of 1001 while the other endpoint had a code of 1010, the logical AND would be 1000 which indicates the line segment lies outside of the window.

On the other hand, if the endpoints had codes of 1001 and 0110, the logical AND would be 0000, and the line could not be trivially rejected.

Cohen-Sutherland Line Clipping

The logical **OR** of the endpoint codes determines if the line is completely inside the window. If the logical OR is zero, the line can be trivially accepted.

Ex:

If the endpoint codes are 0000 and 0000, the logical OR is 0000 - the line can be trivially accepted.

If the endpoint codes are 0000 and 0110, the logical OR is 0110 and the line cannot be trivially accepted.

Cohen-Sutherland Line Clipping Algorithm

The Cohen-Sutherland algorithm uses a divide-and-conquer strategy.

To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each end point of the line segment is then assigned the code of the region in which it lies.

- Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$

- 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.

- 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.

- Examine one of the endpoints, Read 's 4-bit code in order: **Left**-to-**Right**, **Bottom**-to-**Top**.
- When a set bit (1) is found, compute the intersection $\boldsymbol{I}$ of the corresponding window edge with the line from P_1 to P_2 . Replace P_1 with $\boldsymbol{I}$ and repeat the algorithm.

Liang-Barsky Line Clipping

The ideas for clipping line of Liang-Barsky and Cyrus-Beck are the same

The only difference is Liang-Barsky algorithm has been optimized for an upright rectangular clip window

Liang and Barsky have created an algorithm that uses floating-point arithmetic but finds the appropriate end points with at most four computations.

Liang-Barsky Line Clipping

Let $P(x_1,y_1)$, $Q(x_2,y_2)$ be the line which we want to study.

$x = x_1 + (x_2 - x_1)*t = x_1 + dx*t$ and $y = y_1 + (y_2 - y_1)*t = y_1 + dy*t$

- Parametric Equation of the line

Line (x_1, y_1) *to* (x_2, y_2)

Consider t: range from 0 to 1

At Start of the line (x_1, y_1) t=0

At End of the line (x_2, y_2) t=1

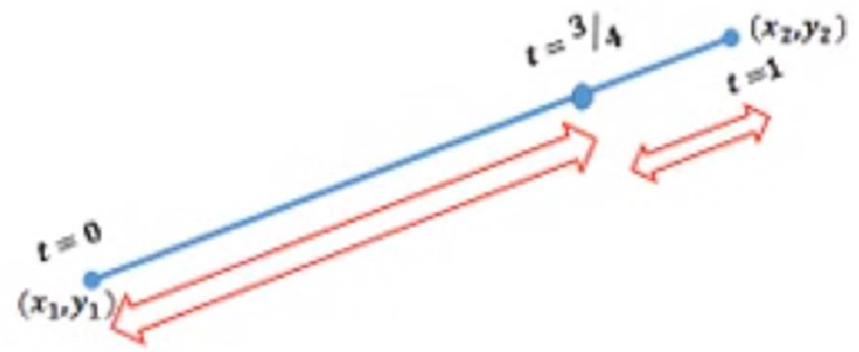
At ${}^{3}/_{4}{}^{th}$ of the line path

$t={}^{3}/_{4}$

The location is

$x={}^{1}/_{4}\,x_1 + {}^{3}/_{4}\,x_2$

$y={}^{1}/_{4}\,y_1 + {}^{3}/_{4}\,y_2$



Liang-Barsky Line Clipping

At time t

x=(1-t). $x_1 + t.x_2$

y=(1-t). $y_1 + t.y_2$

x= $x_1 - x_1.t + t.x_2$

$= x_1 + t(x_2 - x_1)$

$$x = x_1 + t\Delta x$$

y= $y_1 - y_1.t + t.y_2$

$= y_1 + t(y_2 - y_1)$

$$y = y_1 + t\Delta y$$

Parametric Line Equation is

$$x = x_1 + t\Delta x$$
$$y = y_1 + t\Delta y$$

Liang-Barsky Line Clipping Algorithm

$x_w min \le x \le x_w max$
$y_w min \le y \le y_w max$

Sub x ,y value

$x_w min \le x_1 + t\Delta x \le x_w max$
$y_w min \le y_1 + t\Delta y \le y_w max$

$x_1 + t\Delta x \ge x_w min$
$x_1 + t\Delta x \le x_w max$
$y_1 + t\Delta y \ge y_w min$
$y_1 + t\Delta y \le y_w max$

$t\Delta x \ge x_w min - x_1$
$t\Delta x \le x_w max - x_1$
$t\Delta y \ge y_w min - y_1$
$t\Delta y \le y_w max - y_1$

$-t\Delta x \le x_1 - x_w min$
$t\Delta x \le x_w max - x_1$
$-t\Delta y \le y_1 - y_w min$
$t\Delta y \le y_w max - y_1$

$tp_k \le q_k [k = 1,2,3,4]$

$p_1 = -\Delta x \quad q_k = x_1 - x_w min$
$p_2 = \Delta x \quad q_2 = x_w max - x_1$
$p_3 = -\Delta y \quad q_3 = y_1 - y_w min$
$p_4 = \Delta y \quad q_4 = y_w max - y_1$

Liang-Barsky Line Clipping Algorithm

Set $t_{min} = 0$ and $t_{max} = 1$

Calculate the values of tL, tR, tT, and tB (tvalues).

- if $t < t_{min}$ or $t > t_{max}$ ignore it and go to the next edge
- otherwise classify the **t**value as entering or exiting value (using inner product to classify)
- if **t** is entering value set $t_{min} = t$; if **t** is exiting value set $t_{max} = t$

If $t_{min} < t_{max}$ then **draw a line** from (x1 + dx*tmin, y1 + dy*tmin) to (x1 + dx*tmax, y1 + dy*tmax)

If the line crosses over the window, you will see (x1 + dx*tmin, y1 + dy*tmin) and (x1 + dx*tmax, y1 + dy*tmax) are intersection between line and edge.

Liang-Barsky Line Clipping Algorithm

Step 1: Get the line Endpoints (x_1, y_1) to (x_2, y_2)

Step 2: Find $\Delta x, \Delta y, P_1, P_2, P_3, P_4, q_1, q_2, q_3, q_4$

Step 3: Assign t1=0 t2=1

i. if $P_k = 0\ (k = 1,2,3,4)$ $then\ line\ is\ parallel\ to\ the\ window$

ii. if $q_k < 0\ (k = 1,2,3,4)$ $then\ line\ is\ outside\ the\ window$

iii. For non-zero value of P_k

if $P_k < 0$ $then\ find\ t1$
$t1 = Max\left(0, {}^{q_k}/_{P_k}\right)$

else $P_k > 0$ $then\ find\ t2$
$t2 = Min\left(1, {}^{q_k}/_{P_k}\right)$

If $t1 > t2$ $then\ line\ is\ completely\ outside\ -reject$
or
$else\ find\ new\ set\ of\ (x, y)\ if\ t1, t2\ is\ changed$

$$x = x_1 + t\Delta x$$
$$y = y_1 + t\Delta y$$