COMPUTER GAPHICS AND VISUALIZATION

CONTENTS

# 8. 2D Viewing

T. RAGHUVEERA

epgp books

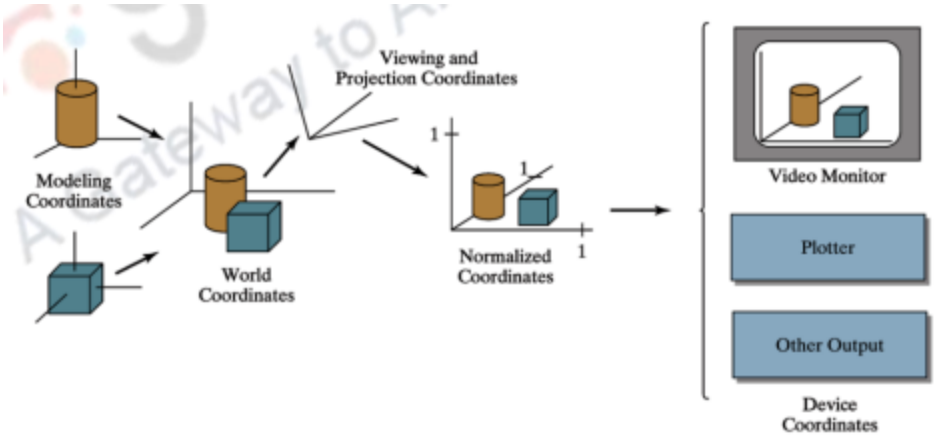
**Objectives:**

- Understand 2D viewing pipeline
- Learn how to perform, window-viewport transformations
- Understand the theory behind clipping in 2D with focus on Point Clipping.

**Discussion:**

**2D Viewing Pipeline:**

Let's start our discussion with 2D viewing pipeline. This involves the transformations performed, right from creation, till the object is finally seen or displayed at a position or a viewing region on a display device like monitor. Each transformation step performed converts data from one coordinate system to the other. From the figure below (Recollect discussion from the first module),




The sequence of steps in the 2D viewing Pipeline are

**MC -> WC->VC-> CC->NVC-> DC**

where

MC – Modeling Coordinates (Modeling or Local Coordinate system)

WC – World coordinates (World Coordinate system)

VC – viewing coordinates (Viewing coordinate system)

CC – Clipping coordinates (Viewing coordinate system)

NVC – Normalized viewing or device independent coordinates (Viewing coordinate system, but normalized)

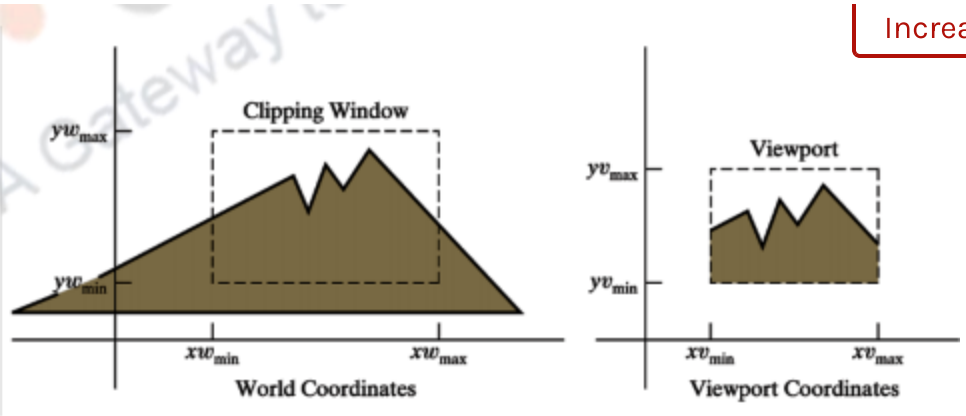
DC – Device coordinates (Device coordinate system specific to every device)

For conversion from MC to WC we perform a transformation called *Modeling* Transformation (MT), from WC to VC we perform *Viewing* Transformation (VT), From VC to NVC we perform *window-viewport* transformation (WV), From NVC to DC we perform *Device coordinate system* transformation (DT). When the object is

first created it assumes MC and a Modeling Transformation is ap
WC, which is in the world coordinate system. Then a viewing coordinate system is set up and data is now converted from WC to VC (recall discussion from the previous module on transformation between coordinate systems). Later VC are sent for clipping, depending on the window or region of interest, to get CC. In many graphics packages, Viewing and Clipping steps are combined to get clipped coordinates straight from world coordinates. Next the CC are converted to NVC, where we simply normalize data in a step called window-Viewport transformation. This step is performed to ensure that the data should be independent of the end display device so that the data can be conveniently mapped to any display device of choice. If this is not done we need to perform a separate transformation for each display device, which is a cumbersome task. Each kind of coordinates belong to a specific coordinate system or reference frame.

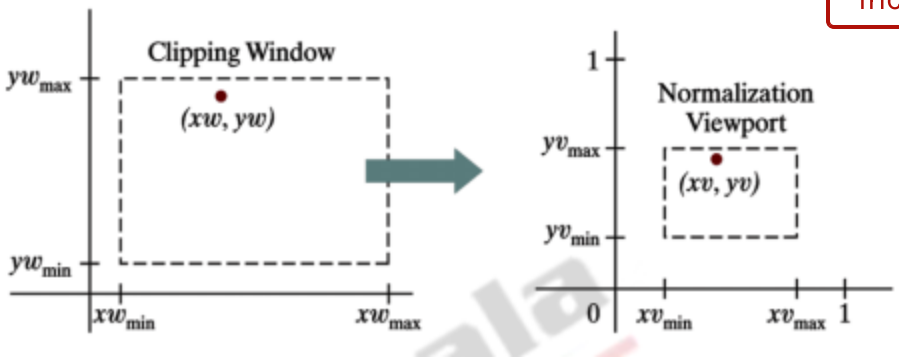
**Window-Viewport Transformation:**

In Computer Graphics terminology, a *window* is a rectangular region of interest, and a *viewport* is a display area chosen on the monitor screen for viewing the contents of the window. In short, *window* is 'what we want to see' and *viewport* is 'where we want to see'. As shown in the figure below, we can define a rectangular region in the world coordinate system, and call that as *window*. The contents of the *window* are to be mapped to a region or a display area chosen as *viewport*. In a way this is, mapping contents between two rectangular regions of different dimensions. *Window* is generally defined in the world coordinate system, while *viewport* is defined in the end device coordinate system. Since we wish to convert data to *device independent* or *normalized* coordinates, we perform *window-viewport* transformation. So in *window-viewport* transformation contents of the window are mapped to normalized coordinates (coordinates between 0 and 1).




For simplicity, both window and viewport are chosen to be rectangular in shape whose boundaries are aligned with the coordinate axes. Because the dimensions of window and viewport are different, we will be performing a scaling like operation. If the window is bigger in size than the viewport, we perform a scale down operation and vice-versa. As you can notice from the figure above, the boundaries of the window and viewport are defined by four infinitely extending lines parallel to the coordinate axes. The window boundaries are defined by the four lines, xwmin, xwmax, ywmin, yWmax. Where xwmin, xwmax define the two vertical boundaries, while, ywmin and ywmax define the two horizontal boundaries of the window. Similarly the viewport boundaries are defined by four infinitely extending lines, xvmin, xvmax, yvmin, yvmax.

Now for a point (xw,yw) in the window, we have to identify an equivalent point in the viewport, say (xv,yv). i.e., for every point in the window, there is an equivalent point in the viewport. The four corner vertices of the window have got their equivalent vertex points in the viewport, which are nothing but the four corner points of the viewport. For the point (xwmin, ywmin) in the window, the equivalent point in the viewport is (xvmin, yvmin) and so on.




For any general point in the window, how do we find an equivalent point in the viewport? As we can see, from the figures above, the percentage horizontal distance of the point in the window from its left boundary, should be exactly equal to the percentage horizontal distance of the equivalent point in the viewport from its left boundary. We can apply the same analogy to the vertical distance of the point from its bottom boundary.

We can represent the percentage horizontal distance of the point from the left boundary as

$$\frac{xw - xw_{min}}{xw_{max} - xw_{min}} \times 100 = \frac{xv - xv_{min}}{xv_{max} - xv_{min}} \times 100$$

After cancelling 100 on both sides we have

$$\frac{xw - xw_{min}}{xw_{max} - xw_{min}} = \frac{xv - xv_{min}}{xv_{max} - xv_{min}}$$

Our aim to compute the values of xv and yv in terms of other variables in the equation. We get

$$xv = xv_{min} + \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} \times (xw - xw_{min})$$

We can rewrite the above equation as

$$xv = xv_{min} + (xw - xw_{min})s_x$$

Where $s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$

Here $s_x$ is a ratio and is also the scaling factor along x-direction. If the horizontal boundary of the window is larger than that of the viewport then $s_x$ <1 (down scaling),

otherwise $s_x$ >1 (up scaling). If both the boundaries are of same leng scaling).

Similarly we can represent the percentage vertical distance of the point from the bottom boundary as

$$\frac{yw - yw_{min}}{yw_{max} - yw_{min}} \times 100 = \frac{yv - yv_{min}}{yv_{max} - yv_{min}} \times 100$$

After cancelling 100 on both sides we have

$$\frac{yw - yw_{min}}{yw_{max} - yw_{min}} = \frac{yv - yv_{min}}{yv_{max} - yv_{min}}$$

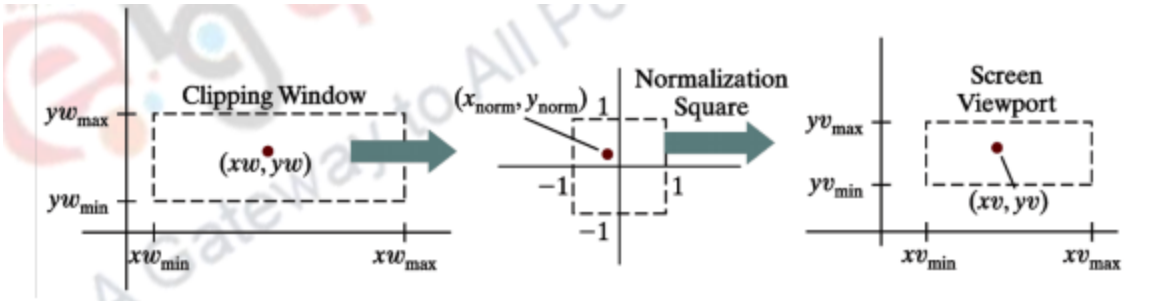
And rearranging the terms as we did before for yv we get,

$$yv = yv_{min} + (yw - yw_{min})s_y$$

Where $s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$

Here sy is a ratio and is also the scaling factor along y-direction. If the vertical boundary of the window is larger than that of the viewport then sy <1 (down scaling), otherwise sy >1 (up scaling). If both the boundaries are of same lengths then sy=1 (no scaling).

It is important to note that the transformation, converts real coordinates to normalized coordinates, which are device independent. Later the device independent coordinates are converted to appropriate screen coordinates as shown in the figure below.



**Example:**

**Problem:** For a point (1, 2) in the window, find the equivalent point in the viewport. Assume window at [(1,1) and (4,4)] and viewport at [(-1, -1) and (1,1)].

**Sol:**

The point (1, 2) is identified as ($xw$, $yw$) respectively. The aim is to compute the equivalent point ($xv$,$yv$).

$xwmin$=1, $xwmax$=5, $ywmin$=1, $ywmax$=5,
$xvmin$=-1, $xvmax$=1, $yvmin$=-1, $yvmax$=1,
Now compute $s_x$ using the relation derived above

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$s_x = \frac{1-(-1)}{5-1} = ½$$

Now compute $s_y$ using the relation derived above $s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$

$$s_y = \frac{1-(-1)}{5-1} = ½$$

Now compute $xv$ using the relation below

$$xv = xv_{min} + (xw - xw_{min})s_x$$

$$xv = -1 + \frac{(1-1)1}{2} = -1$$

Now compute $xv$ using the relation below

$$yv = yv_{min} + (yw - yw_{min})s_y$$

$$yv = -1 + \frac{(2-1)1}{2} = -\frac{1}{2}$$

The equivalent point for (1,2) is (-1, -1/2)

**********

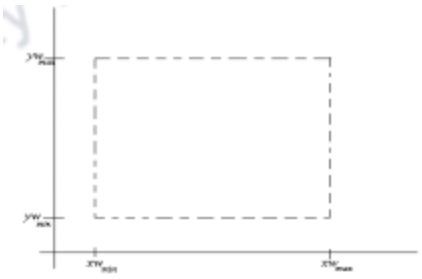
## 2D Clipping Algorithm:



In 2D clipping, we choose a rectangular region of interest in the world coordinate system and call that as window or clip window or clip rectangle. The contents inside the window are of interest to us and so to be clipped. So here clipping means select the contents that are inside the clip window region. It is cumbersome and less efficient to consider performing transformations on the regions that are not of interest to us, instead we can consider the inside of the clip window and perform any transformations on that. Thus clipping is 'selecting what is of interest to us' and **not** 'discarding what is not of interest to us'.

For better understanding and simplicity we choose the clip rectangle to be aligned such that its sides are parallel to the coordinate axes. In the most general case we can choose the clip rectangle to be oriented in any direction in 2D space. In the extreme case, the shape of the clip window region need not be rectangular, instead, it can be of any shape of choice.

The region against which the contents are clipped is called 'clip window'. Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is considered as the 'clipping algorithm'. Usually a clipping region is a rectangle in *standard position*. These boundary edges typically correspond to a normalized square in which the x and y values range either from 0 to 1 or from -1 to 1.



Clipping transformation can be performed either in the world coordinate system or in the viewport coordinate system. Clipping in the World coordinate system involves selecting the contents inside the clip window region and apply transformations only

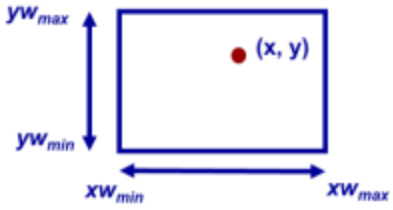
on them. In viewport clipping, the whole of the contents from the to the viewport and then clipping against the viewport. It is thus more convenient to perform clipping in the world coordinate system.

The types of clipping that we are going to learn are

- Point clipping
- Line clipping
- Polygon Clipping
- Text Clipping

**Point Clipping:**

In point clipping, points that are inside the rectangular region are selected for display and points that are outside the region are discarded. Now the question is, how do we know whether a point is inside or outside a given rectangular region? For this we should perform a test as shown.




$$xw_{min} \leq x \leq xw_{max}$$
$$yw_{min} \leq y \leq yw_{max}$$

From the above two equations we can infer that if x and y satisfy the above equations, the point is an inside point, so can be selected for display, otherwise it is an outside point so can be discarded. In the above two equations, there are 4 inequalities,

$$xw_{min} \leq x, \quad \text{- (1)}$$
$$x \leq xw_{max}, \quad \text{- (2)}$$
$$yw_{min} \leq y, \quad \text{- (3)}$$
$$y \leq yw_{max} \quad \text{- (4)}$$

Increase Font Size

The *x* coordinate should satisfy two inequalities, $xw_{min} \leq x$ , $x \leq xw_{max}$, which says that the x coordinate should be between the two vertical boundaries $xw_{min}$ and $xw_{max}$, while the *y* coordinate should satisfy two other inequalities, $yw_{min} \leq y$, $y \leq yw_{max}$ which means that the y-coordinate should lie between the two horizontal boundaries $yw_{min}$ and $yw_{max}$ . A point (x,y) is said to be an inside point, only if all the 4 inequalities are satisfied by the point. Let's check how? Say for example, the first inequality, $xw_{min} \leq x$ is not satisfied by the x-coordinate, instead it satisfied the relation, $xw_{min} > x$, it means that, x coordinate is onto the left of the left boundary of the window, and so we can infer that the point is outside the clip window region and so can be discarded. Even if one of the inequalities is not satisfied by the point, the point can be safely ignored.

**********

**Example:**

**Problem:** Check the position of point (2,1) with respect to the clip window at [(2,2), (4,4)].
**Sol:** of course, the point (2,1) is outside the clip window and so can be discarded.
Discussion: Treating the given point as (x,y) and $xw_{min}$=2, $xw_{max}$=4, $yw_{min}$=2, $yw_{max}$=4, Except the 3rd inequality, the other inequalities are satisfied by the point. So as per the discussion above, the point is an outside point.

**********

## Summary:

- Understood the steps in 2D viewing pipeline
- Learnt about window-viewport transformation
- Also learnt the idea behind clipping, and in particular point clipping

**you can view video on 2D Viewing**