Computer Graphics

Unit 2 – Part 3

-By Manjula. S

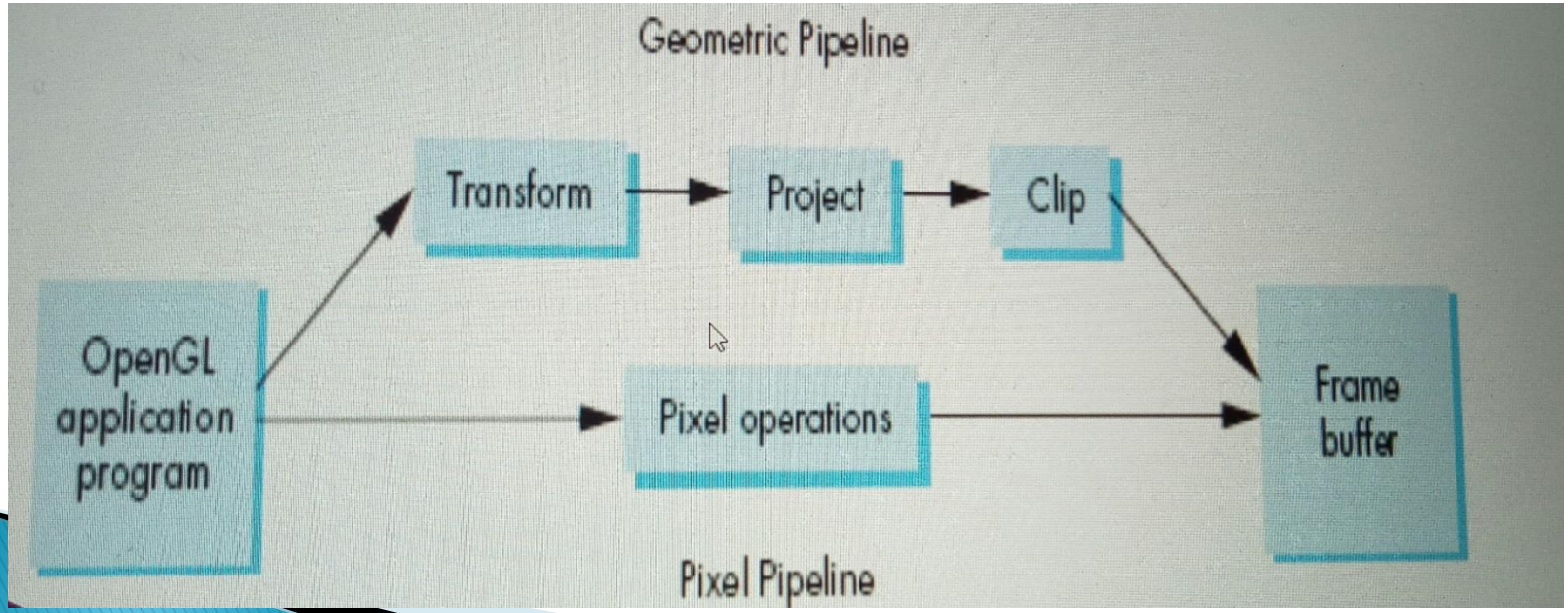
veiwing

- Viewing is a type of graphics functions which enables the programmer to specify the view facilities such as top view, front view, left side view, right side view, zoom in and zoom out.
- Determines the appearance of the object on the output device.
- If the programmer does not specify a view explicitly, default one(orthogonal view) is considered.
- Viewing can be
  - Orthographic view
  - 2D view
  - Matrix mode

2D viewing Pipeline

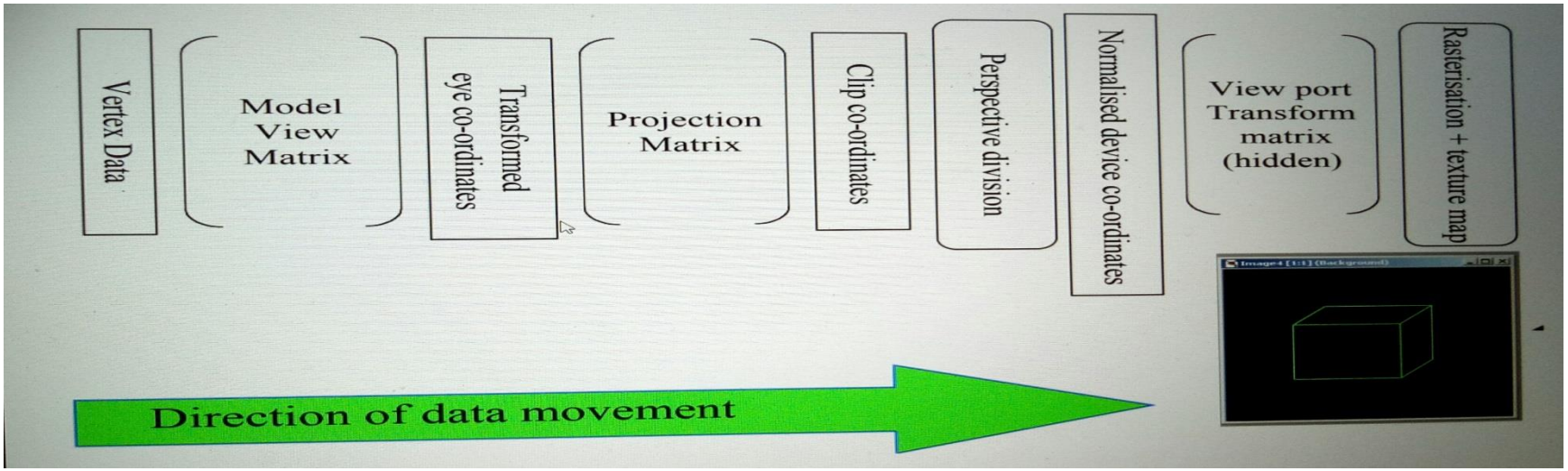
OpenGL supports the following primitives

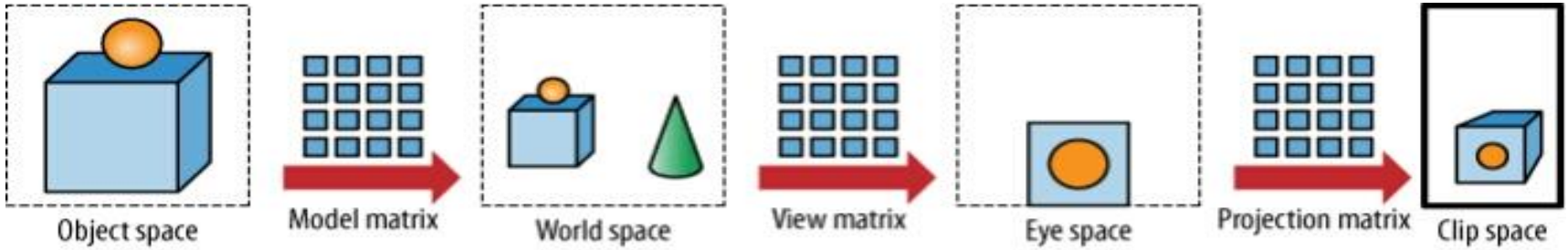
- Geometric primitives Points, line segments, polygons, curves, surfaces
- Image OR raster primitives



-By Manjula. S

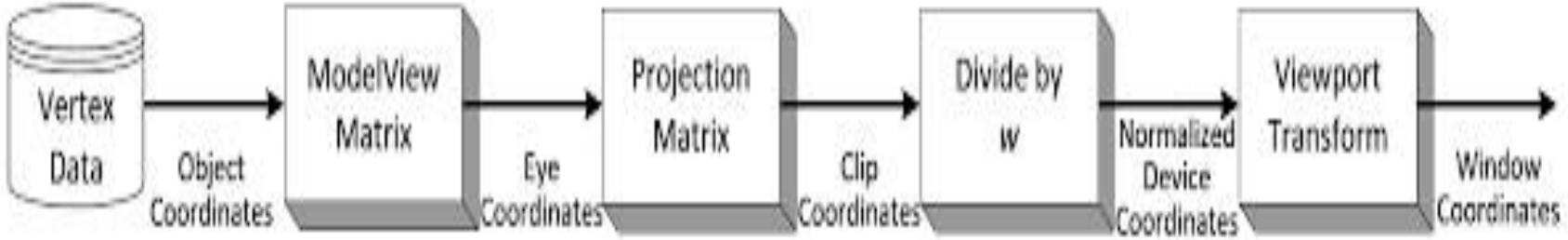
OpenGL Matrix Operations

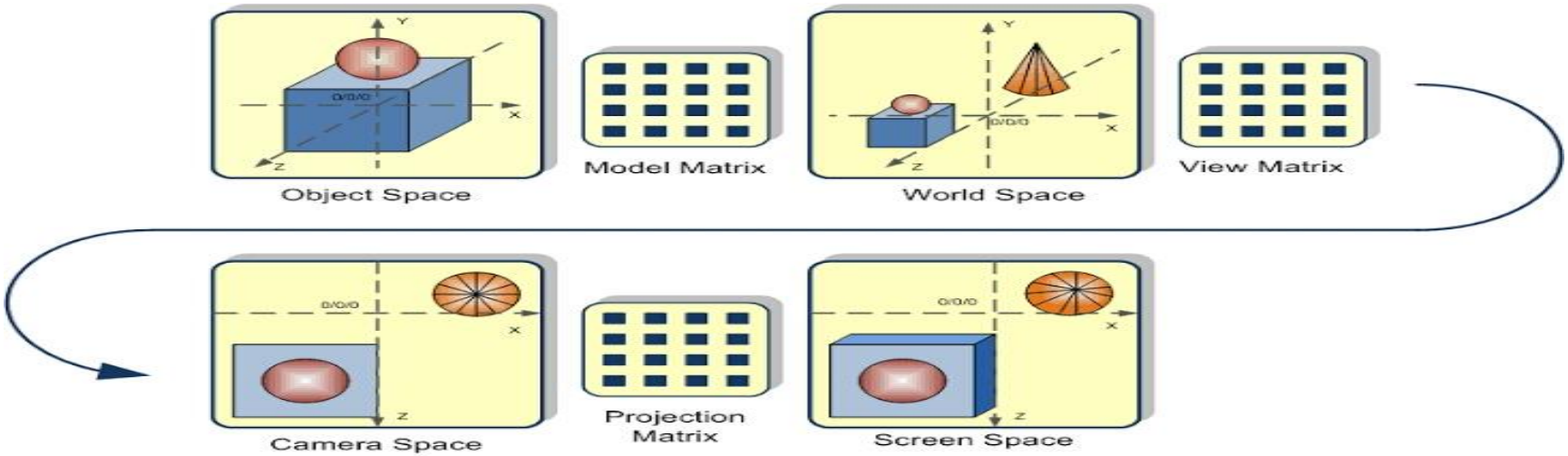




-By Manjula. S

OpenGL Matrix Operations





-By Manjula. S

OpenGL Matrix Operations

- **Modelview matrix**: combines modelling and viewing transforms
- **Projection matrix**: projects 3-D viewing coordinates onto image plane

Combining All Three

$$\underbrace{\begin{bmatrix} 1.25 & 0 & 0 & 0 \\ 0 & 1.667 & 0 & 0 \\ 0 & 0 & -1.1333 & -10.667 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{\text{projection}} \quad \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -14 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{view}} \quad \underbrace{\begin{bmatrix} 0.9107 & -0.2440 & 0.3333 & 0 \\ 0.3333 & 0.9107 & -0.2440 & 0 \\ -0.2440 & 0.3333 & 0.9107 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{model}}$$

$$\underbrace{\begin{bmatrix} 0.9107 & -0.2440 & 0.3333 & 0 \\ 0.3333 & 0.9107 & -0.2440 & 0 \\ -0.2440 & 0.3333 & 0.9107 & -14 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{modelview}}$$

$$\underbrace{\begin{bmatrix} 1.1384 & -0.3050 & 0.4167 & 0 \\ 0.5556 & 1.5178 & -0.4067 & 0 \\ 0.2766 & -0.3778 & -1.0321 & 5.2 \\ 0.2440 & -0.3333 & -0.9107 & 14 \end{bmatrix}}_{\text{modelview-projection}}$$

Matrix-by-matrix multiplication is associative so
PVM = P (V M) = (P V) M

OpenGL keeps V and M "together" because eye-space is a convenient space for lighting

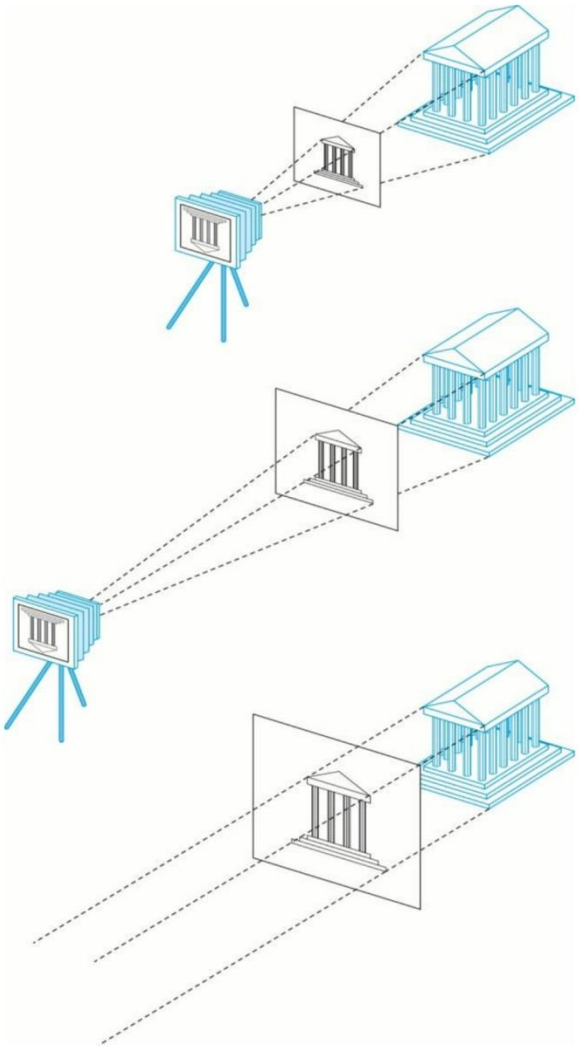
-By Manjula. S

Normalization

- Rather than derive a different projection matrix for each type of projection, we can convert all projections to orthogonal projections with the default view volume
- This strategy allows us to use standard transformations in the pipeline and makes for efficient clipping

Orthographic View

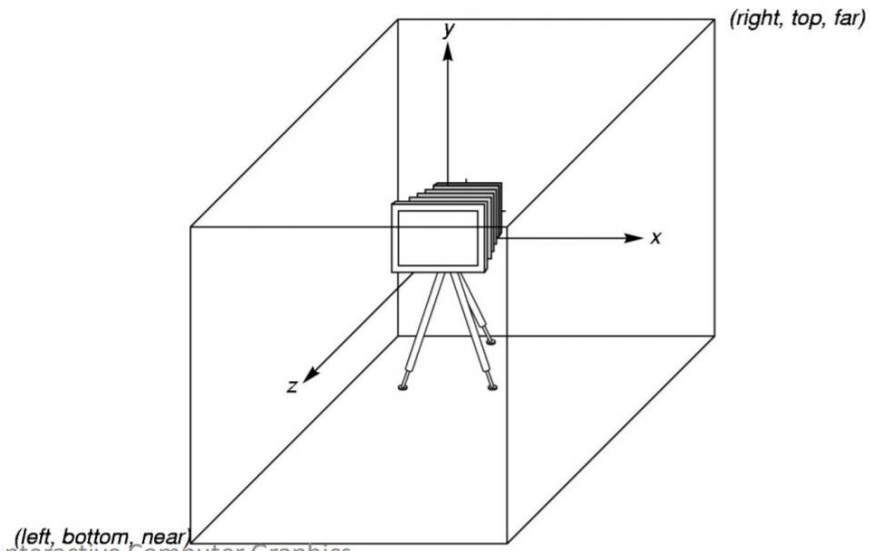
- Simplest and OpenGL's default view is orthographic projection
- Can be obtained in the synthetic camera model by placing the camera at an infinitely far distance from the object.
- The projections(lines) are perpendicular (orthogonal) to the projection plane and hence the projection is called as orthogonal projection/viewing.



-By Manjula. S

Orthographic View

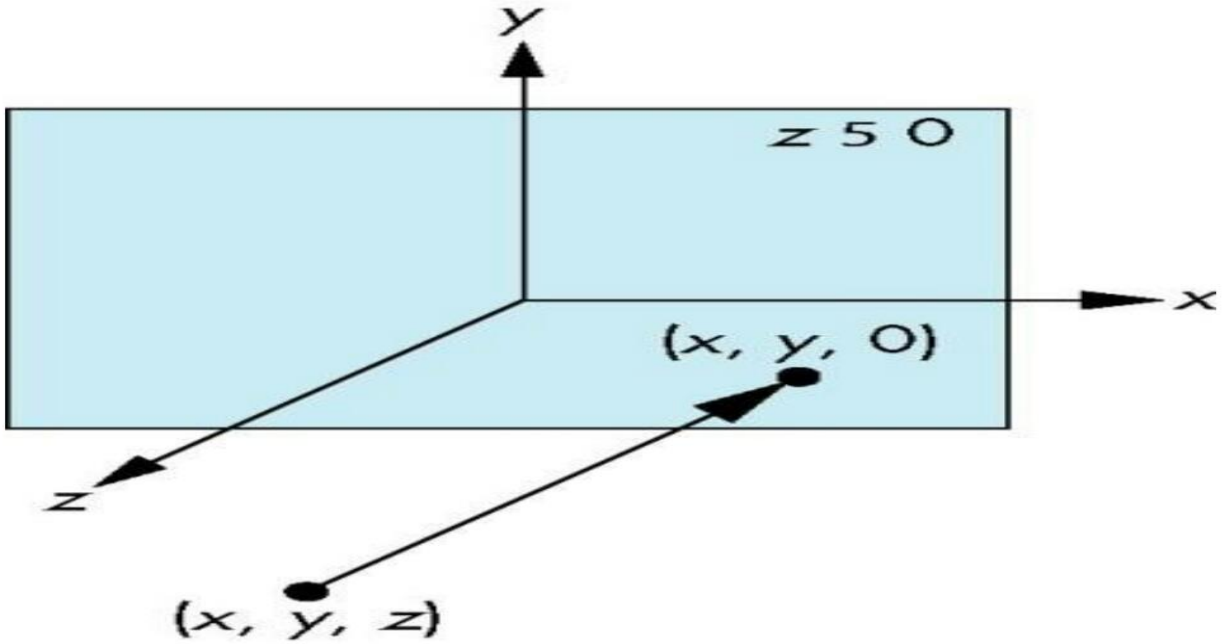
- OpenGL places a camera at the origin in object space pointing in the negative z direction
- The default viewing volume is a box centered at the origin with a side of length 2.



-By Manjula. S

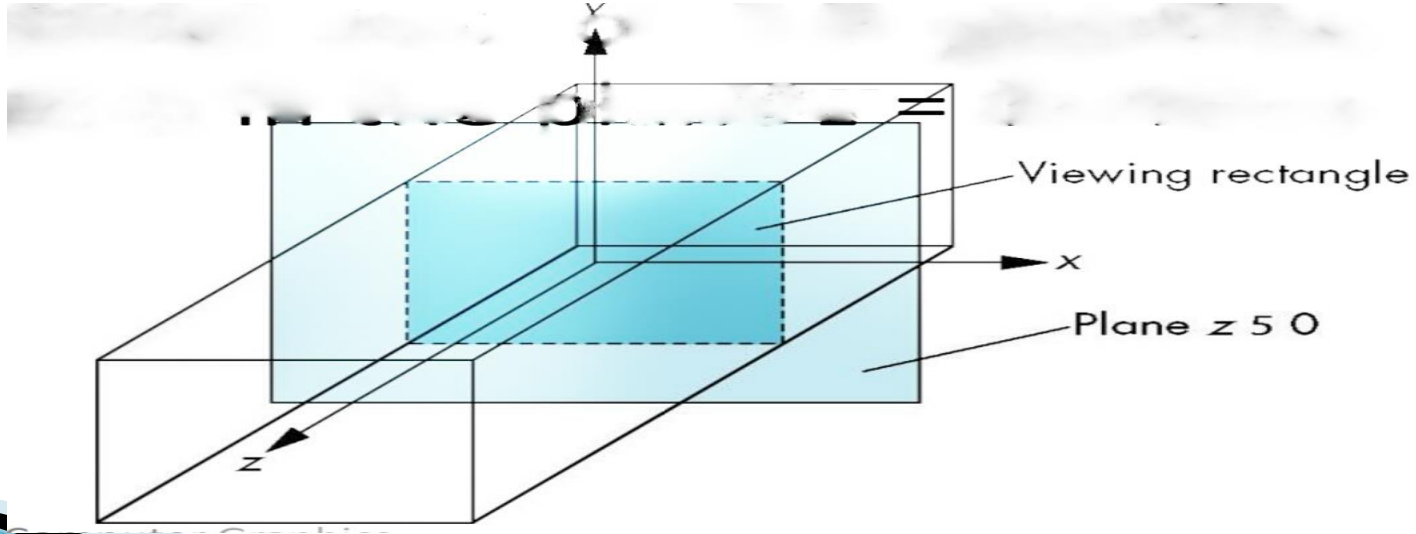
2D Orthographic View

- Orthographic projection takes a point (x,y,z) and projects it into the point (x,y,0)
- In 2-D viewing, with all vertices in plane z=0, a point and its projection both are same.



2D Orthographic View

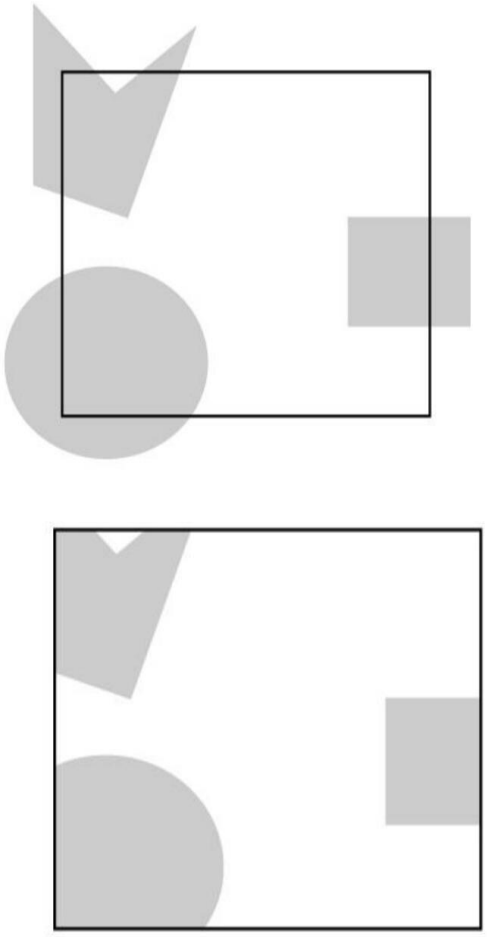
- If viewing volume is not specified, OpenGL uses its default, a 2 x 2 x 2 cube, with the origin in the center. In terms of 2-D plane, the bottom-left corner is at(-1.0, -1.0), and the upper-right corner is at (1.0, 1.0).
- 2-D graphics view is a special case of 3-D graphics. Hence the viewing rectangle is in the plane z = 0 within 3-D viewing volume:



-By Manjula. S

2D viewing

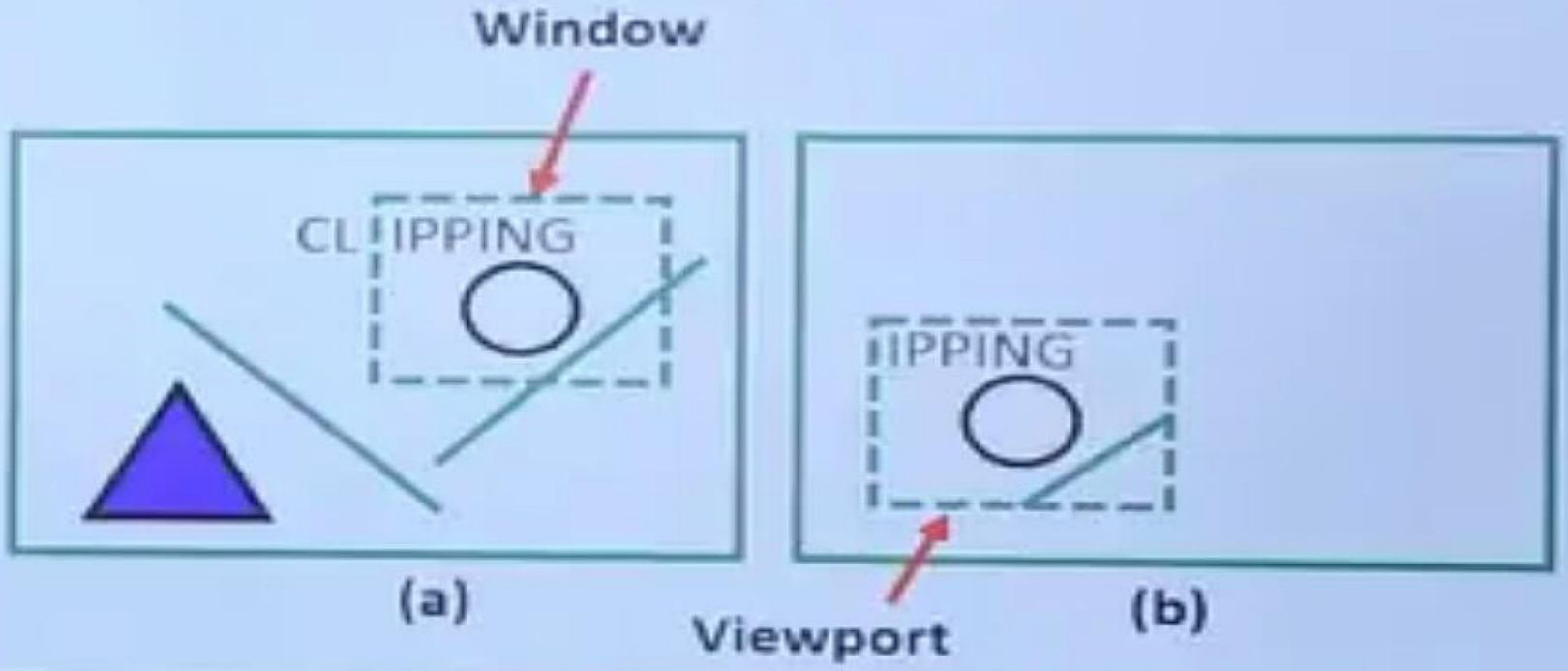
- 2D viewing : is based on taking a rectangular area of 2D world and transferring its contents to the display .
- The area of the world of which the image is to be taken is known as the viewing rectangle or clipping rectangle .
- Objects inside the rectangle are in the image; objects outside are clipped out and are not displayed .



-By Manjula. S

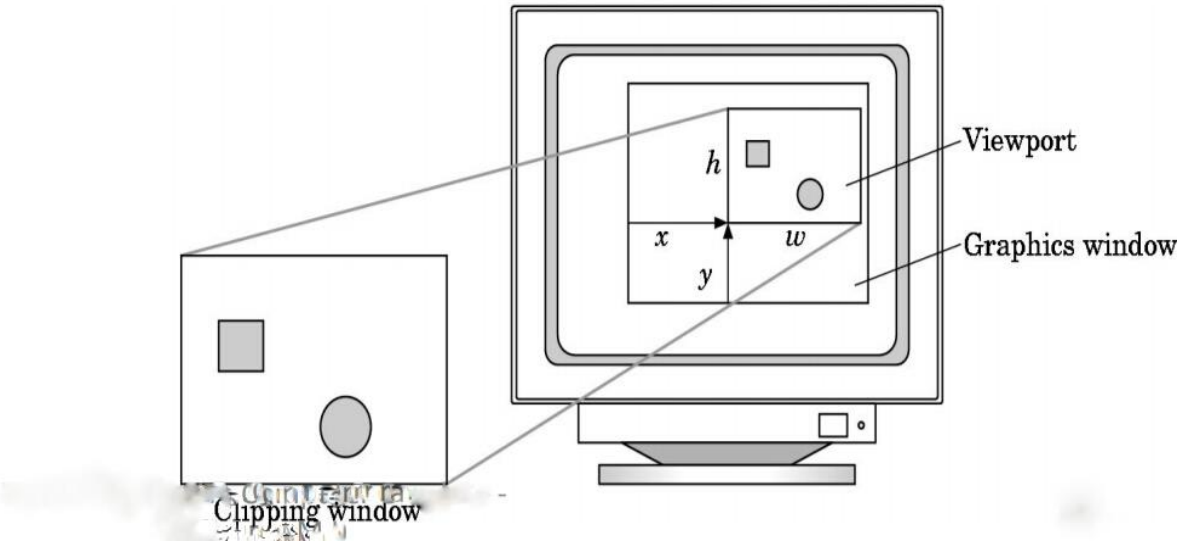
Window to Viewport Transformation

- By defining a closed boundary or window the enclosed portion of a world coordinate scene is clipped against the window boundary and the data of the clipped portion is extracted for mapping to a separately defined region known as viewport.
- While window selects a part of the scene, viewport displays the selected part at desired location on the display area.

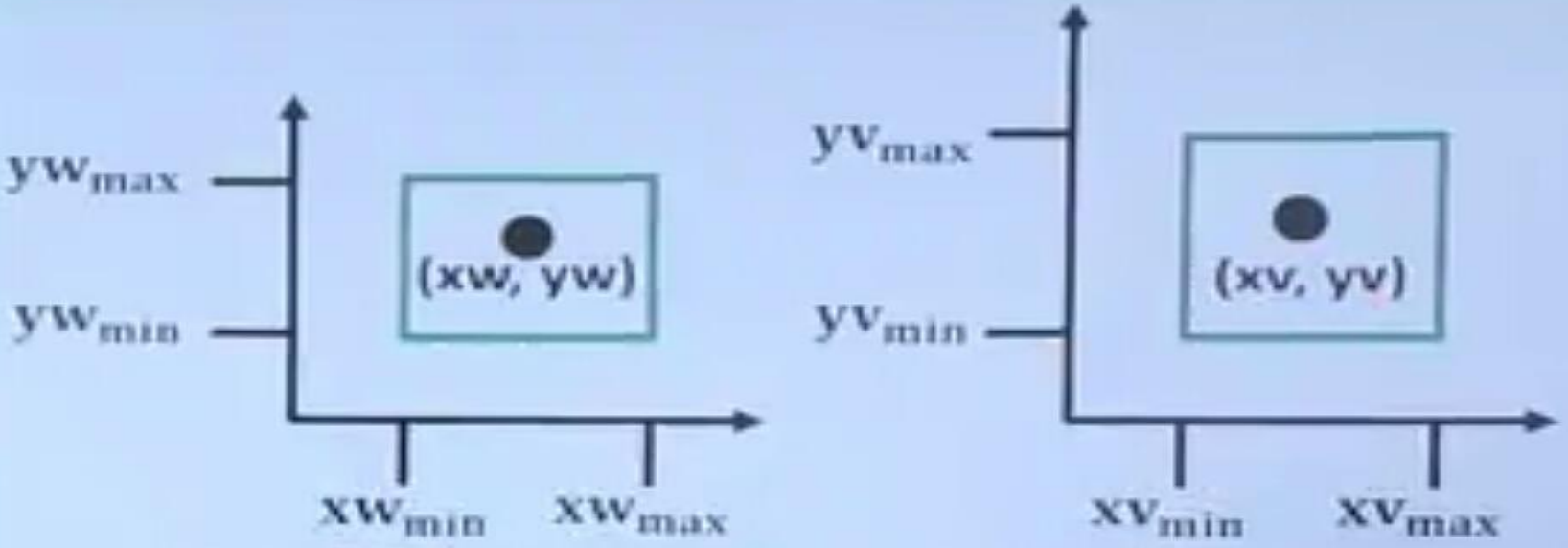


Aspect Ratio and Viewports

- ***Aspect ratio*** : ratio of the rectangle's width to its height.
- ***Viewport*** : Rectangular area of the display window.
- By default, it is the entire window, but can be set to any smaller size in pixels through function.



Window to Viewport Transformation



(a) Window in object space (b) Viewport in device space

To maintain the same relative placement of the point in the viewport as in the window,

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

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

-By Manjula. S

Window to Viewport Transformation

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

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

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

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

If we consider $s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$ and $s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$

Thus we have $xv = xv_{min} + (xw - xw_{min})\, s_x$

And $yv = yv_{min} + (yw - yw_{min})\, sy$

-By Manjula. S

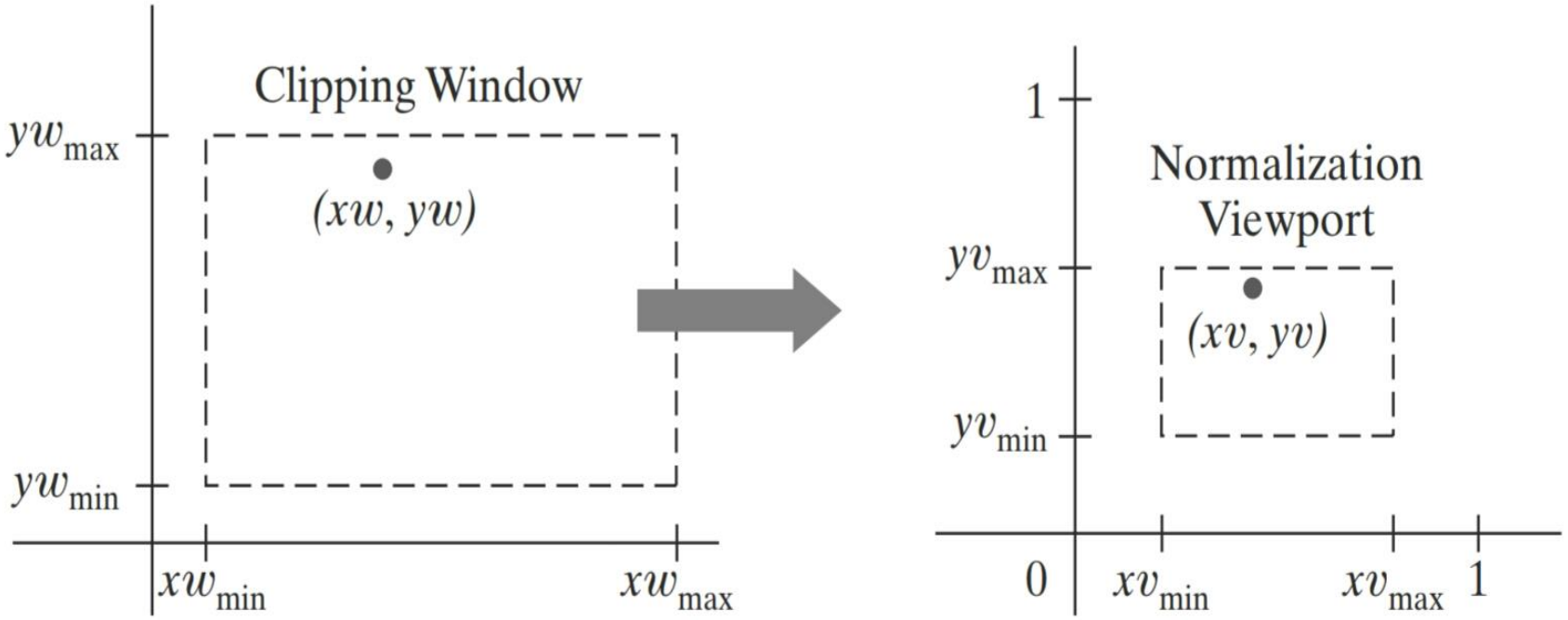


FIGURE 6

A point (xw, yw) in a world-coordinate clipping window is mapped to viewport coordinates (xv, yv), within a unit square, so that the relative positions of the two points in their respective rectangles are the same.

-By Manjula. S

To transform the world-coordinate point into the same relative position within the viewport, we require that

$$\frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}} = \frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}}$$
$$\frac{yv - yv_{\min}}{yv_{\max} - yv_{\min}} = \frac{yw - yw_{\min}}{yw_{\max} - yw_{\min}} \quad (2)$$

Solving these expressions for the viewport position (xv, yv), we have

$$xv = s_x xw + t_x$$
$$yv = s_y yw + t_y \quad (3)$$

where the scaling factors are

$$s_x = \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}$$
$$s_y = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}} \quad (4)$$

and the translation factors are

$$t_x = \frac{xw_{\max} xv_{\min} - xw_{\min} xv_{\max}}{xw_{\max} - xw_{\min}}$$
$$t_y = \frac{yw_{\max} yv_{\min} - yw_{\min} yv_{\max}}{yw_{\max} - yw_{\min}} \quad (5)$$

-By Manjula. S

Because we are simply mapping world-coordinate positions into a viewport that is positioned near the world origin, we can also derive Equations 3 using any transformation sequence that converts the rectangle for the clipping window into the viewport rectangle. For example, we could obtain the transformation from world coordinates to viewport coordinates with the following sequence:

1. Scale the clipping window to the size of the viewport using a fixed-point position of (xw_{min}, yw_{min}).
2. Translate (xw_{min}, yw_{min}) to (xv_{min}, yv_{min}).

The scaling transformation in step (1) can be represented with the two-dimensional matrix

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & xw_{\min}(1 - s_x) \\ 0 & s_y & yw_{\min}(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

where s_x and s_y are the same as in Equations 4. The two-dimensional matrix representation for the translation of the lower-left corner of the clipping window to the lower-left viewport corner is

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & xv_{\min} - xw_{\min} \\ 0 & 1 & yv_{\min} - yw_{\min} \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

And the composite matrix representation for the transformation to the normalized viewport is

$$\mathbf{M}_{\text{window, normviewp}} = \mathbf{T} \cdot \mathbf{S} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

-By Manjula. S

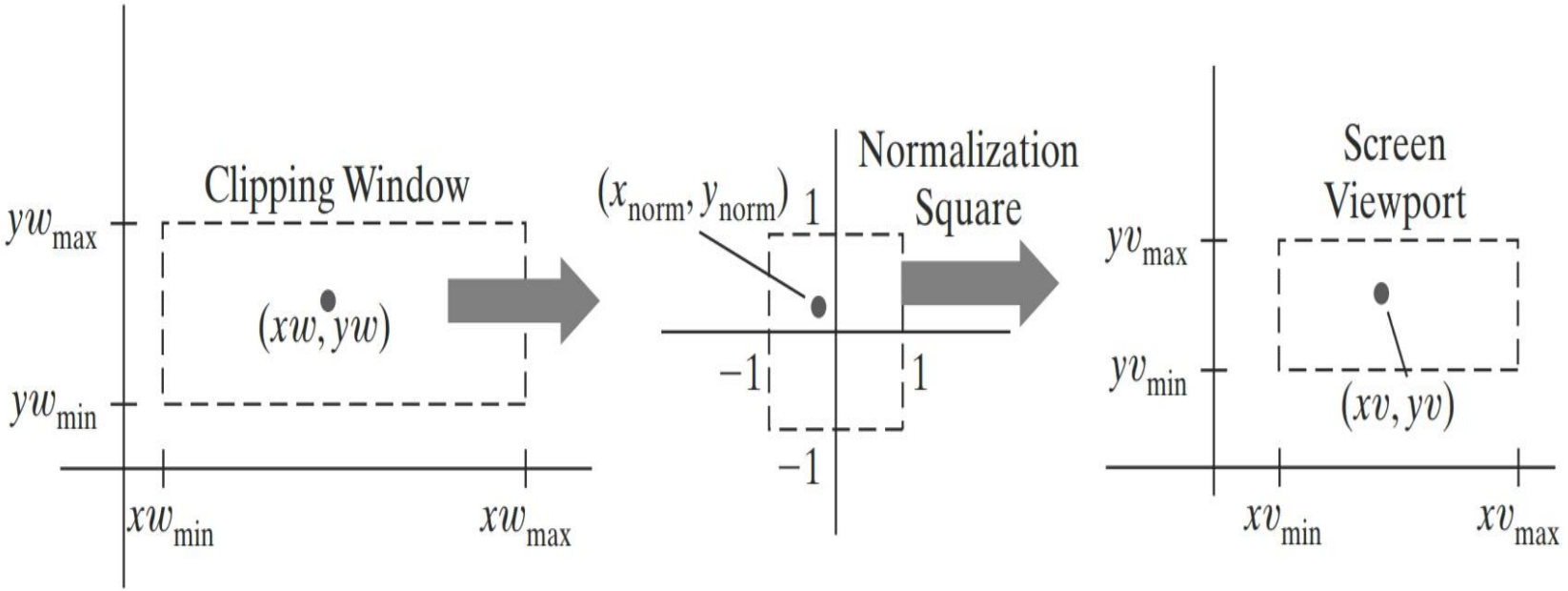


FIGURE 7

A point (xw, yw) in a clipping window is mapped to a normalized coordinate position (x_{norm}, y_{norm}), then to a screen-coordinate position (xv, yv) in a viewport. Objects are clipped against the normalization square before the transformation to viewport coordinates occurs.

Making these substitutions in the expressions for t_x, t_y, s_x, and s_y, we have

$$\mathbf{M}_{\text{window, normsquare}} = \begin{bmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Similarly, after the clipping algorithms have been applied, the normalized square with edge length equal to 2 is transformed into a specified viewport. This time, we get the transformation matrix from Equation 8 by substituting -1 for $xw_{\min}$ and $yw_{\min}$ and substituting $+1$ for $xw_{\max}$ and $yw_{\max}$:

$$\mathbf{M}_{\text{normsquare, viewport}} = \begin{bmatrix} \frac{xv_{\max} - xv_{\min}}{2} & 0 & \frac{xv_{\max} + xv_{\min}}{2} \\ 0 & \frac{yv_{\max} - yv_{\min}}{2} & \frac{yv_{\max} + yv_{\min}}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

The last step in the viewing process is to position the viewport area in the display window. Typically, the lower-left corner of the viewport is placed at a coordinate position specified relative to the lower-left corner of the display window. Figure 8 demonstrates the positioning of a viewport within a display window.

View Functions in OpenGL

- Viewing functions enable the programmer to specify views such as front view, top view, side view. Also provide zoom in and zoom out facilities.
- Example :

```
glViewport();
glMatrixMode(GL_PROJECTION);
glMatrixMode(GL_MODELVIEW);
gluOrtho2D();
```

View Functions in OpenGL

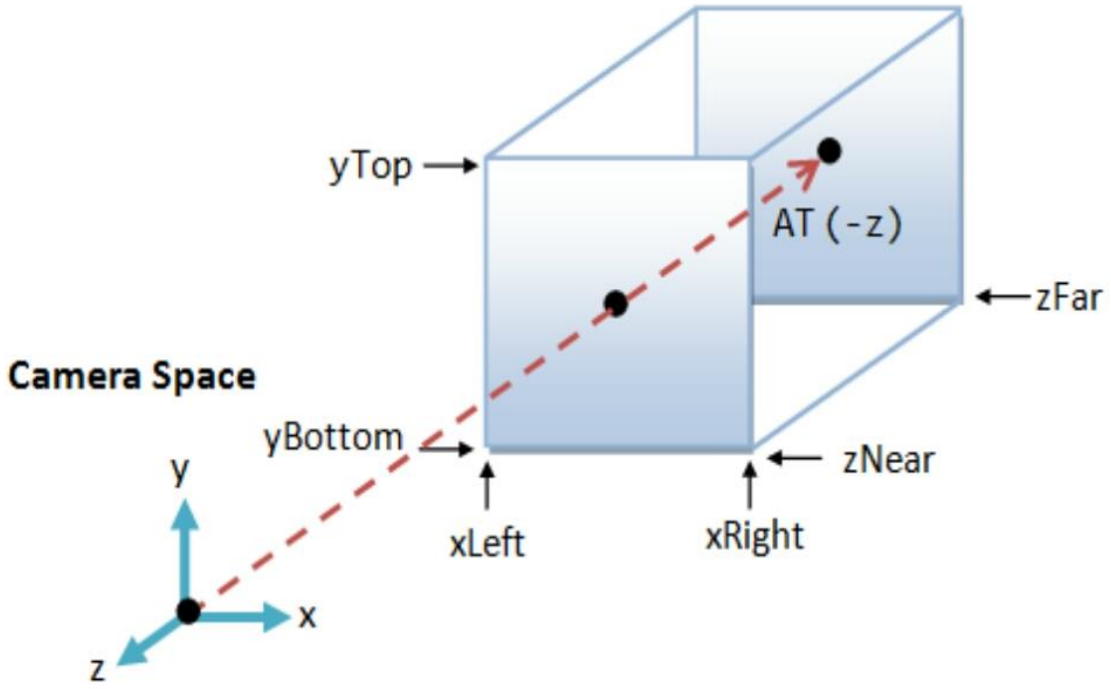
- In OpenGL, an orthographic projection with a right parallel piped viewing volume is specified via the following

void glOrtho(Gldouble left, Gldouble right, Gldouble bottom, Gldouble top, Gldouble near, GLdouble far)

Sees only those objects in the volume specified by the viewing volume. It can also include objects that are behind the camera!

OpenGL Orthogonal Viewing

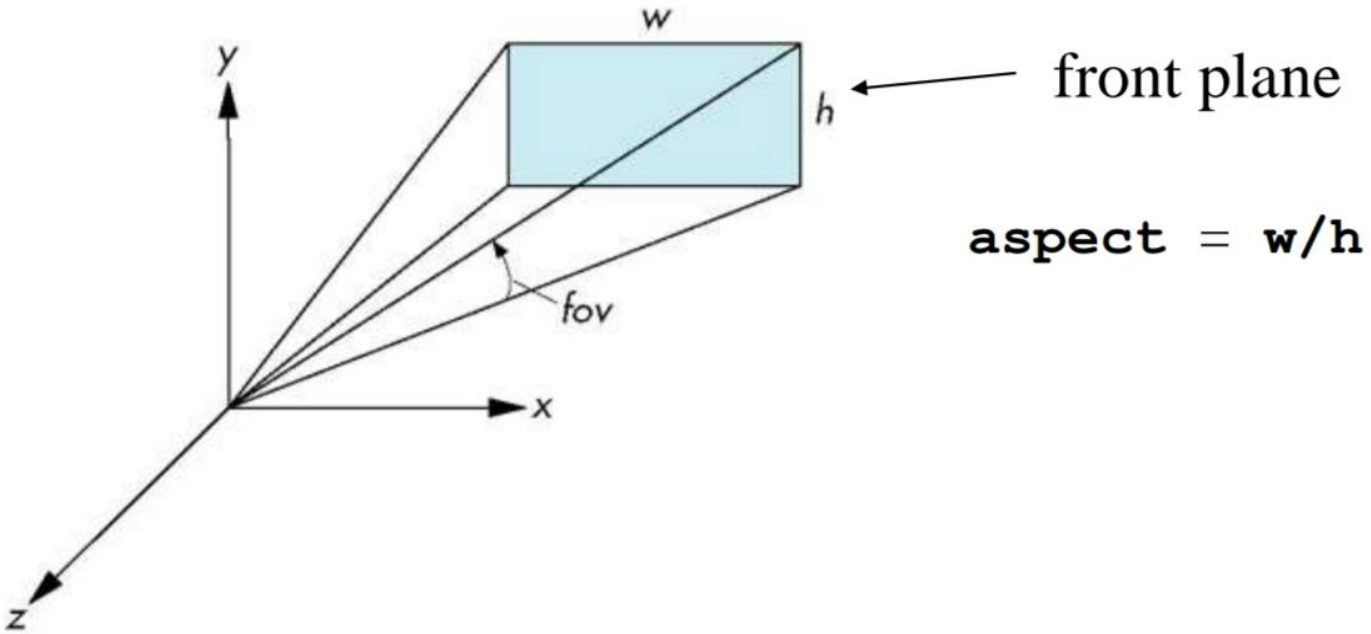
```
glOrtho(left,right,bottom,top,near,far)
```



Orthographic Projection: Camera positioned infinitely far away at $z = \infty$

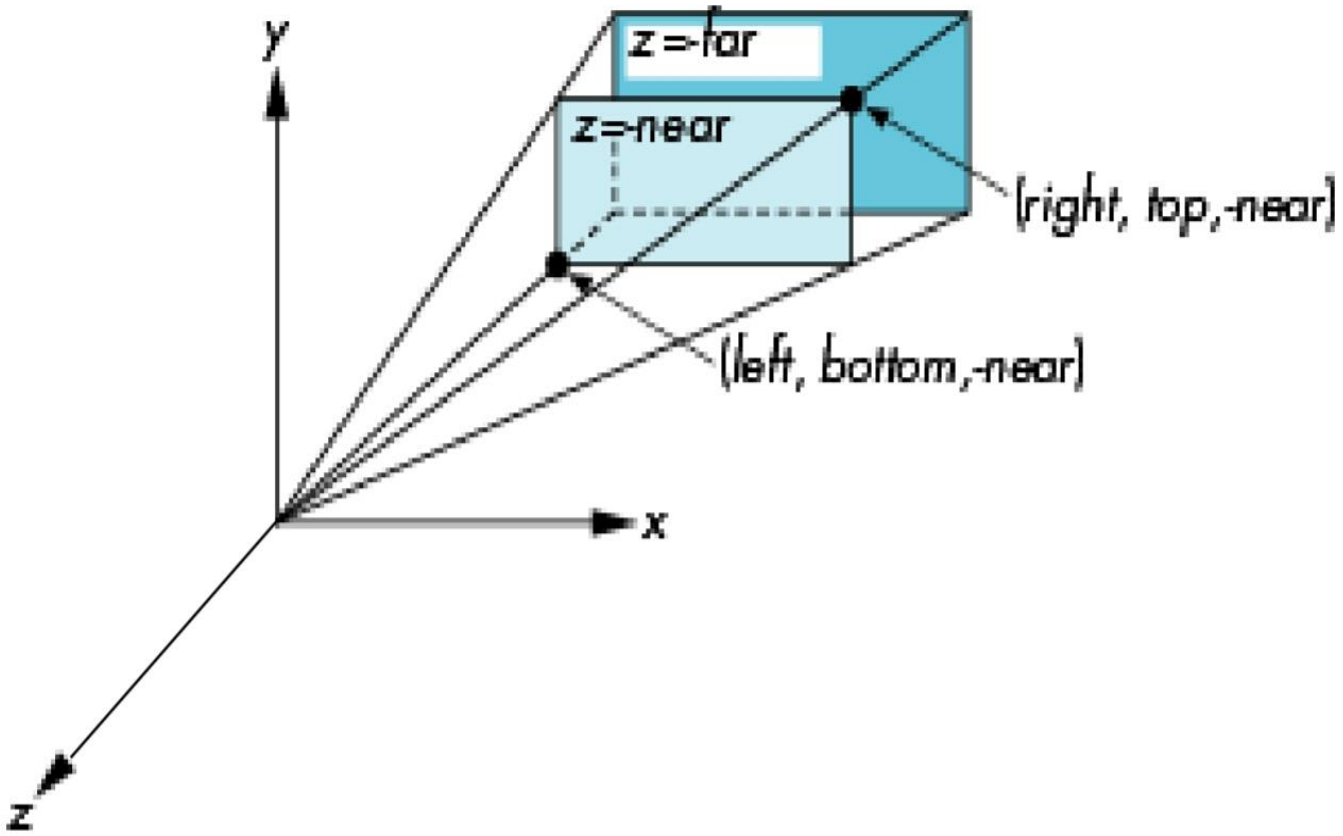
Using Field of View

- With **glFrustum** it is often difficult to get the desired view
- **gluPerpective(fovy, aspect, near, far)** often provides a better interface



OpenGL Perspective

```
glFrustum(left,right,bottom,top,near,far)
```



View Functions in OpenGL

- Objects that straddle the edges of the rectangle are partially visible in the image.
- If using a 3-D volume seems strange in a two dimensional application, the function

void gluOrtho2D(GLdoub1e left, GLdouble right, GLdouble bottom, GLdouble top)

in the utility library may make the program more consistent with its 2-D structure. This function is equivalent to glOrtho with near and far set to -1.0 and 1.0, respectively.

Thank You

-By Manjula. S