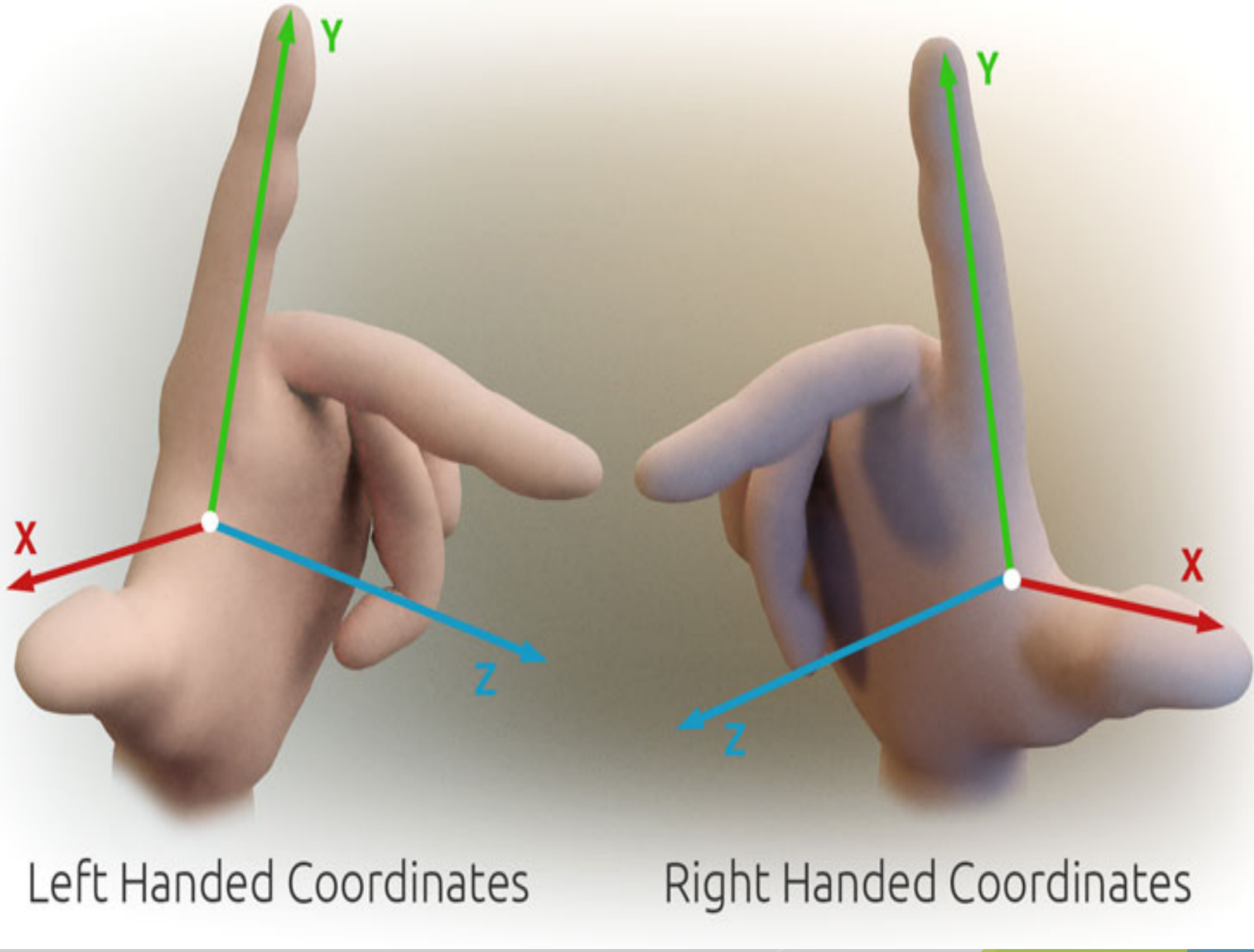
Drawing and Coordinate Systems

Coordinate System "Handedness

- In a 2-D coordinate system the X axis generally points from left to right, and the Y axis generally points from bottom to top.
- (Although some windowing systems will have their Y coordinates going from top to bottom.)
- When we add the third coordinate, Z, we have a choice as to whether the Z-axis points into the screen or out of the screen:

- **Z is coming out of the page**
  +Counterclockwise rotations are positive
  if we rotate about the X axis : the rotation Y->Z is positive
  if we rotate about the Y axis : the rotation Z->X is positive
  if we rotate about the Z axis : the rotation X->Y is positive
  Left Hand Coordinate System (LHS)

- **Z is going into the page**
  Clockwise rotations are positive
  if we rotate about the X axis : the rotation Y->Z is positive
  if we rotate about the Y axis : the rotation Z->X is positive
  if we rotate about the Z axis : the rotation X->Y is


Y
X
Z
Left Handed Coordinates
Y
X
Z
Right Handed Coordinates

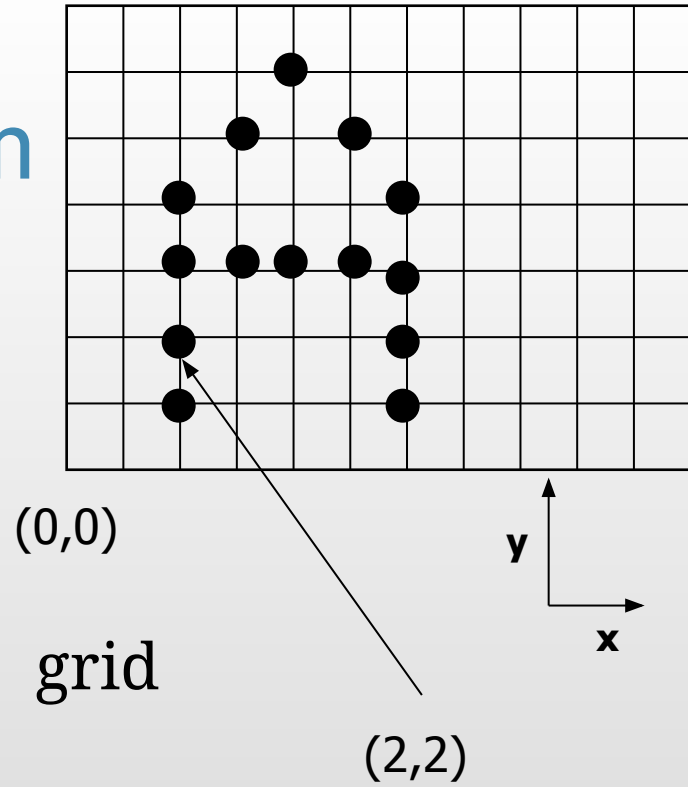
Coordinate Systems

- Screen Coordinate system
- World Coordinate system
- World window
- Viewport
- Window to viewport mapping

MODEL/WORLD WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

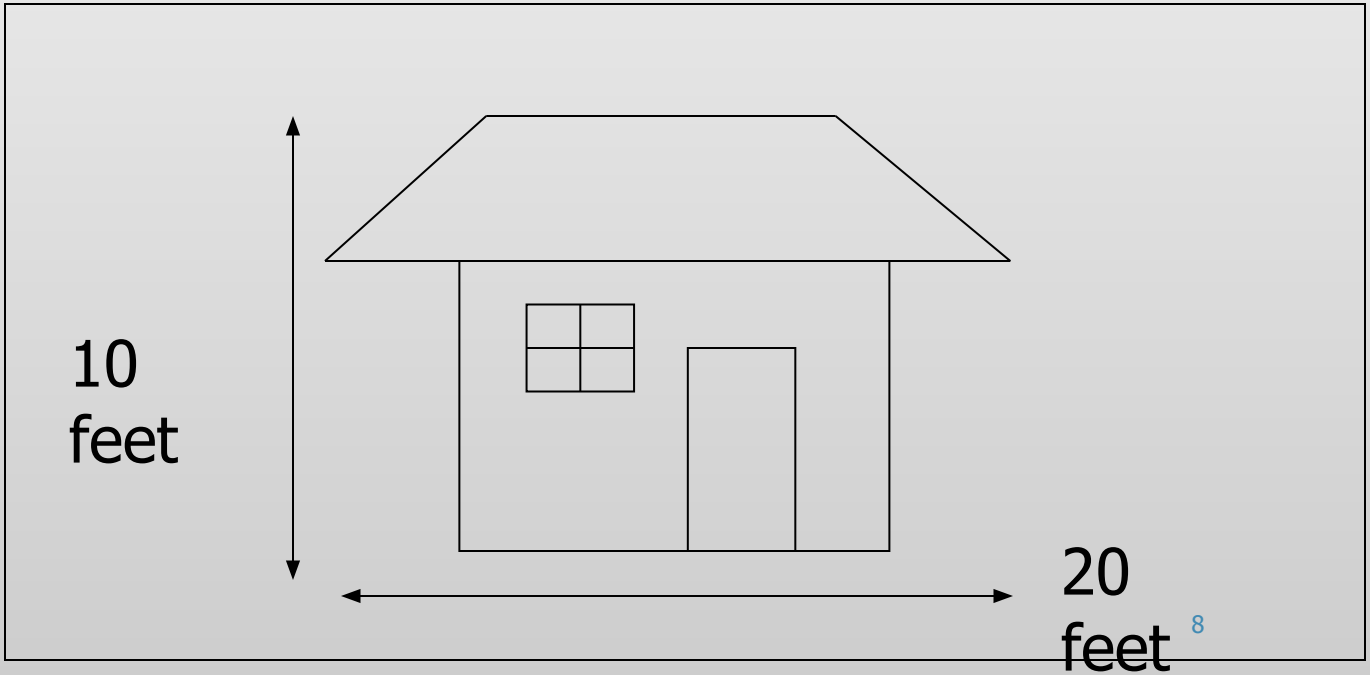
- A world-coordinate area selected for display is called a **window.**
- An area on a display device to which a window is mapped is called a **viewport.**
- The window defines what is to be viewed; the viewport defines where it is to be displayed.

Screen Coordinate System

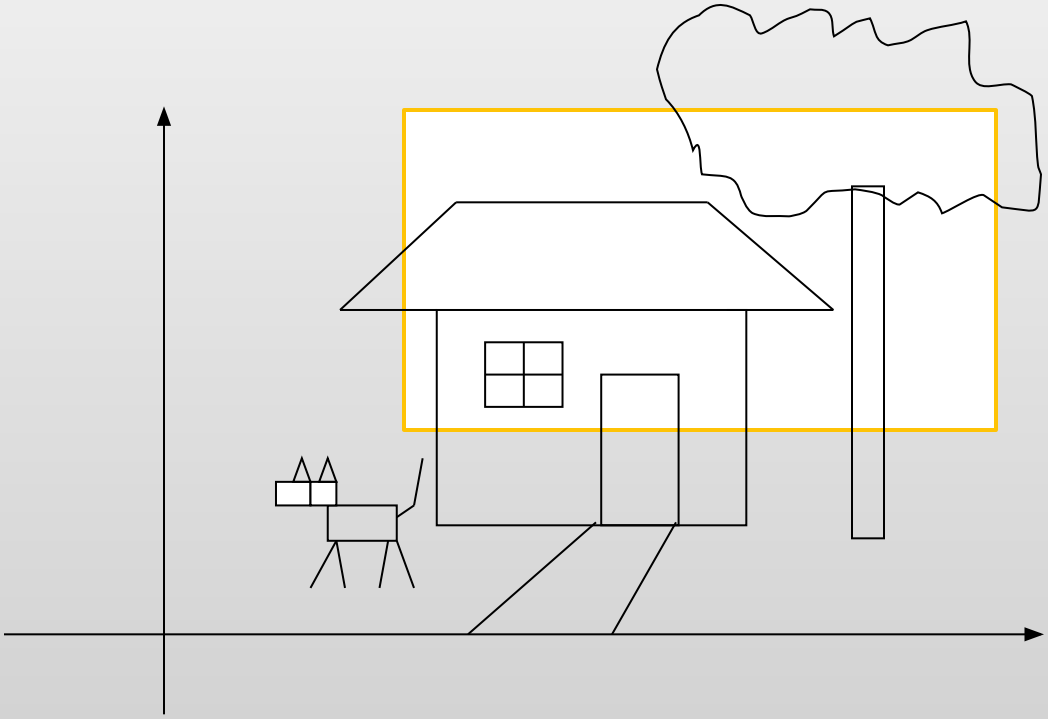


- 2D Regular Cartesian Grid
- Origin (0,0) at lower left corner (OpenGL convention)
- Horizontal axis – x
  Vertical axis – y
- Pixels are defined at the grid intersections
- This coordinate system is defined relative to the display window origin
  (OpenGL: the lower left corner of the window)

World Coordinate System

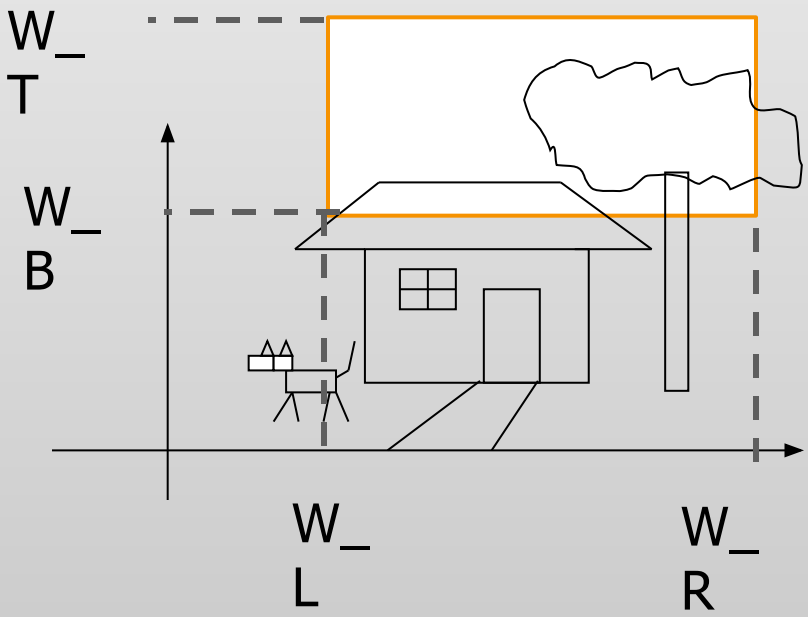


Define a world window



World Window

- World window – a rectangular region in the world that is to be displayed



Define by

W_L, W_R, W_B, W_T

Use OpenGL command:

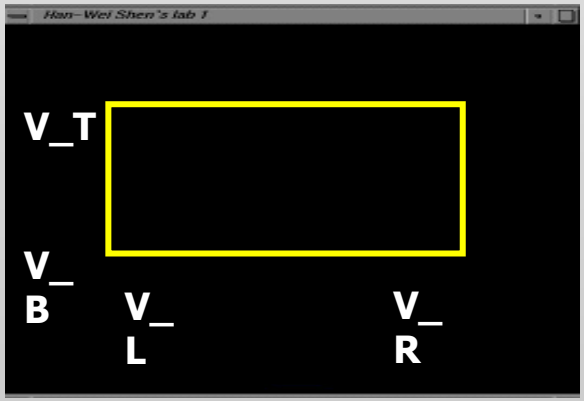
gluOrtho2D(left, right, bottom, top)

Viewport

- The rectangular region in the screen for displaying the graphical objects defined in the world window
- Defined in the screen coordinate system

glViewport(int left, int bottom, int (right-left), int (top-bottom));

call this function before drawing (calling glBegin() and glEnd())

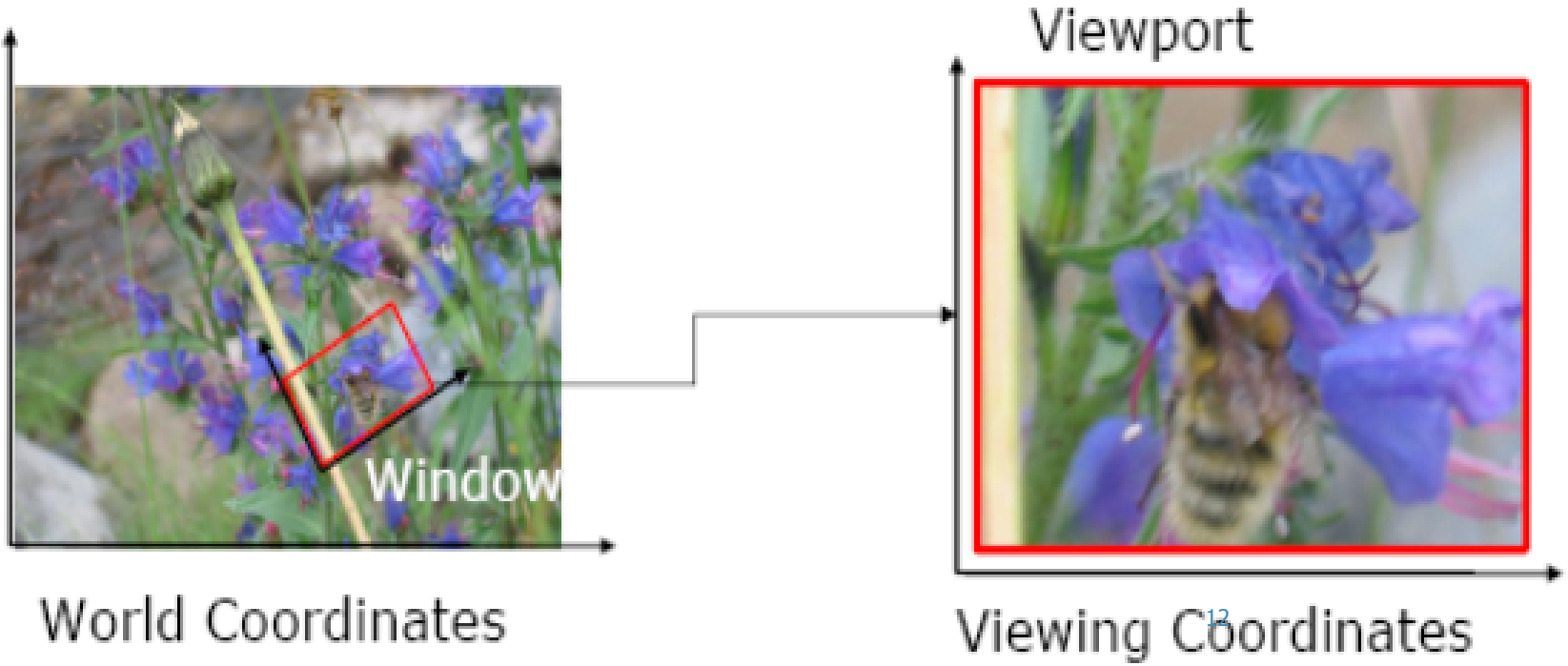


World coordinates to Viewing coordinates

Window to Viewport.

Window: A region of the scene selected for viewing (also called *clipping window*)

Viewport: A region on display device for mapping to window



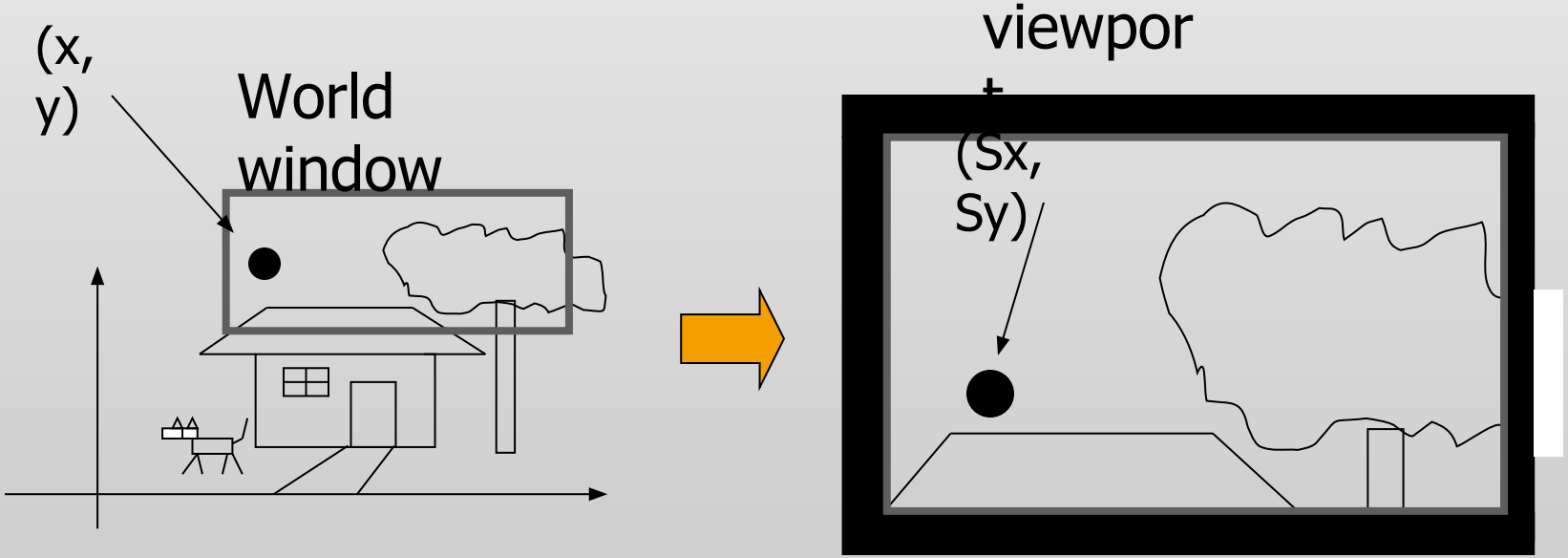
To draw in world coordinate system

- Three tasks need to be done
  - Define a rectangular world window

    (call an OpenGL function)
  - Define a viewport (call an OpenGL function)
  - Perform window to viewport mapping

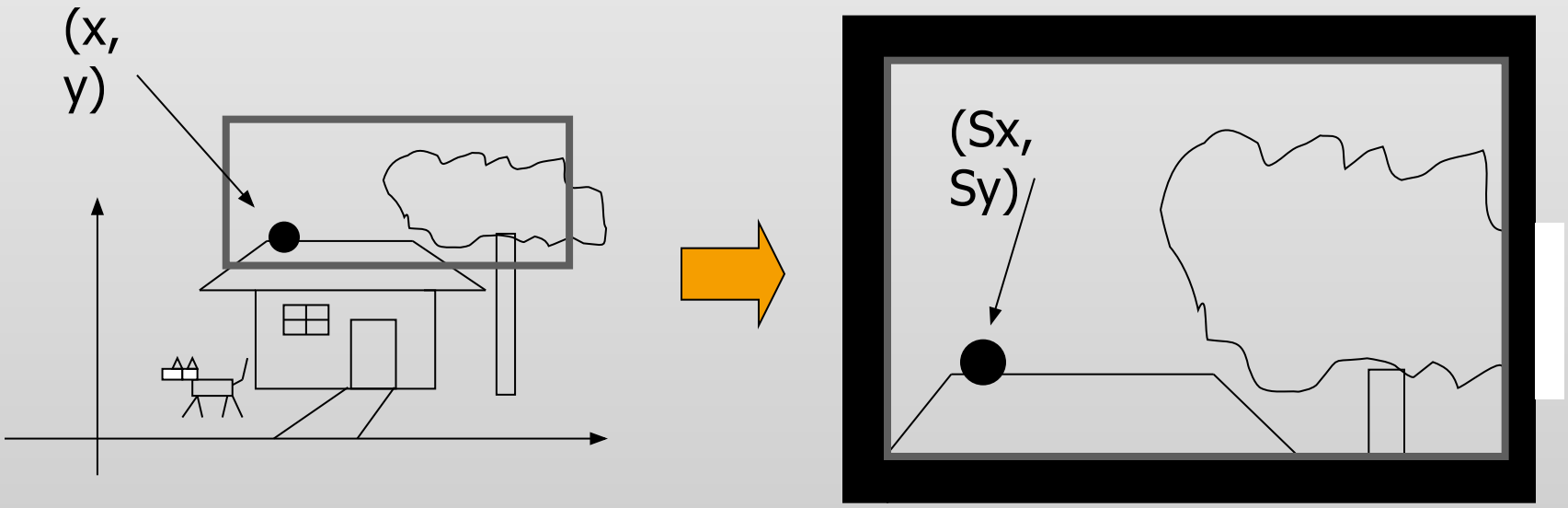
Window to viewport mapping

- The objects in the world window will then be drawn onto the viewport



Window to viewport mapping

- How to calculate (sx, sy) from (x,y)?



Window to viewport mapping

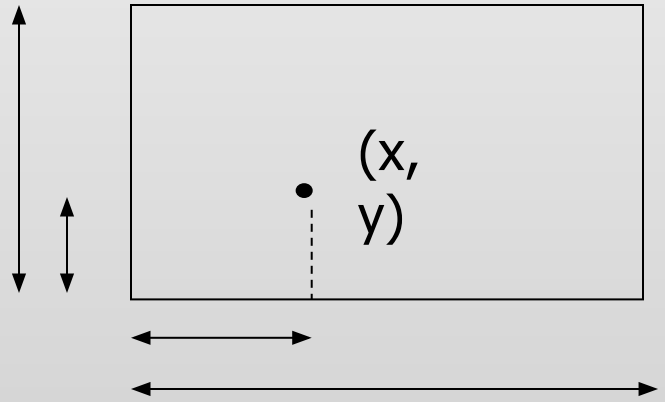
Before calling gluOrtho2D(), you need to have the following two lines of code –

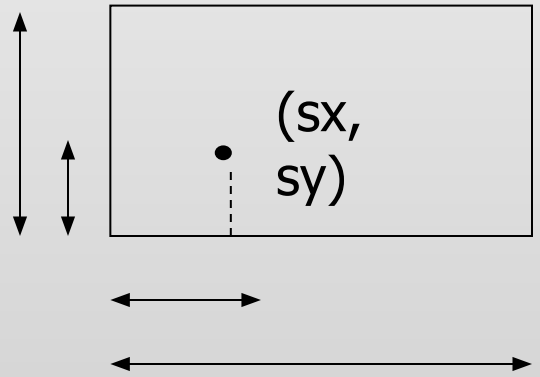
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

- Things that are given:
  - The world window (W_L, W_R, W_B, W_T) (l,r,b,t)
  - The viewport (V_L, V_R, V_B, V_T) (l,b,w,h)
  - A point (x,y) in the world coordinate system
- Calculate the corresponding point (sx, sy) in the screen coordinate system

Window to viewport mapping

- Basic principle: the mapping should be proportional

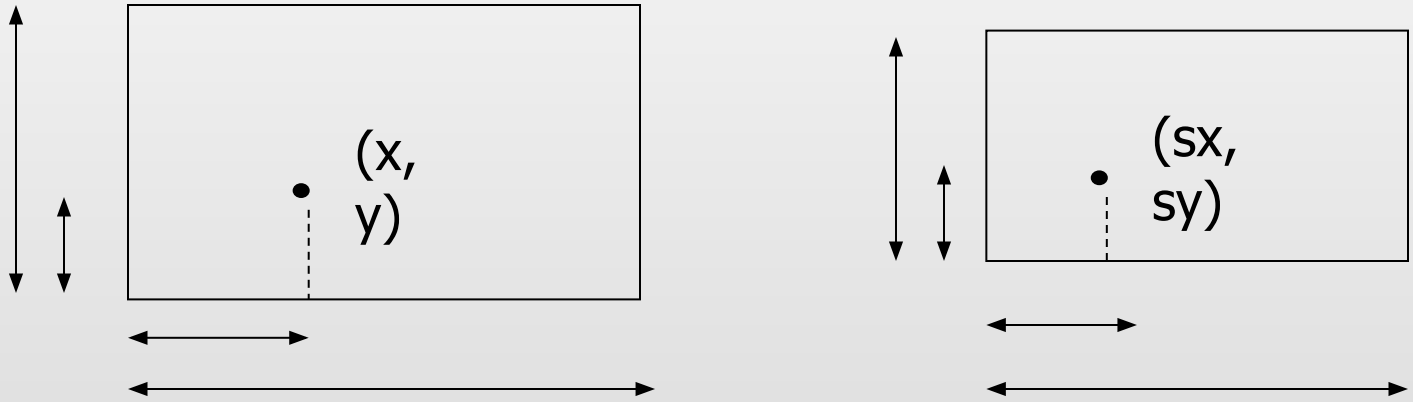




$$(x - W\_L) / (W\_R - W\_L) = (sx - V\_L) / (V\_R - V\_L)$$

$$(y - W\_B) / (W\_T - W\_B) = (sy - V\_B) / (V\_T - V\_B)$$

Window to viewport mapping



$$(x - W\_L) / (W\_R - W\_L) = (sx - V\_L) / (V\_R - V\_L)$$

$$(y - W\_B) / (W\_T - W\_B) = (sy - V\_B) / (V\_T - V\_B)$$

$$sx = x * (V\_R-V\_L)/(W\_R-W\_L) - W\_L * (V\_R - V\_L)/(W\_R-W\_L) + V\_L$$

$$sy = y * (V\_T-V\_B)/(W\_T-W\_B) - W\_B * (V\_T-V\_B)/(W\_T-W\_B) + V\_B$$

Example

- What is (Sx,Sy) for point (3.4,1.2) in world coordinates if:

$$W = (W.L, W.R, W.B, W.T) = (0,4,0,2)$$

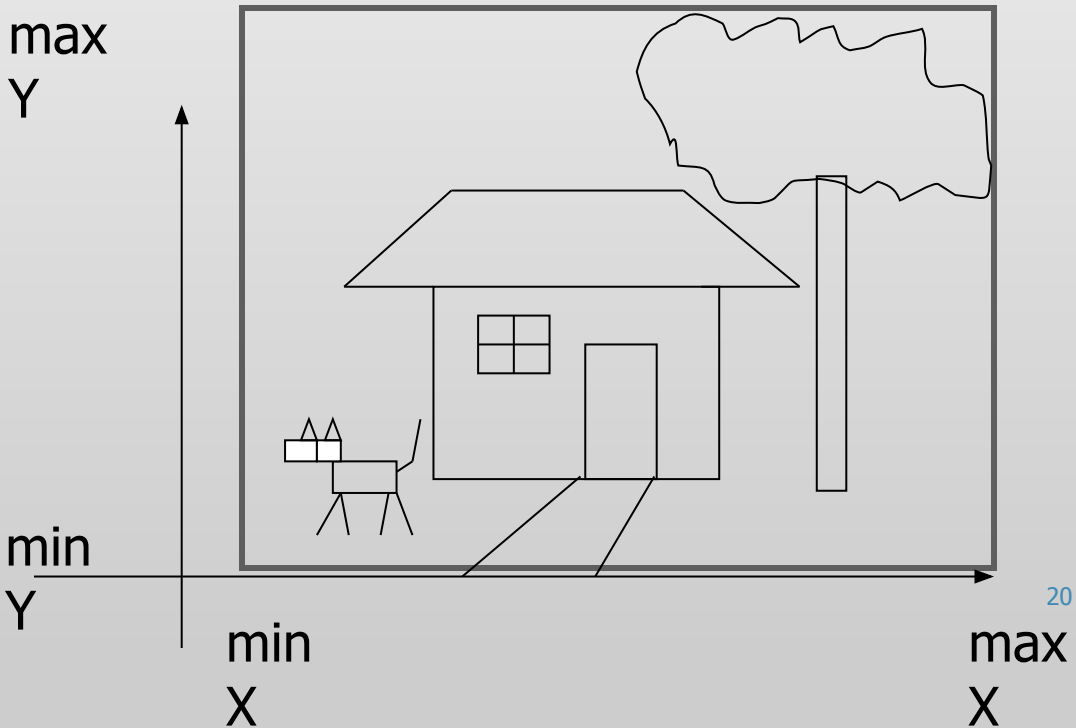
$$V = (V.L, V.R, V.B, V.T) = (60,380,80,240)$$

$$Sx = 80x + 60 = 332 \qquad Sy = 80x + 80 = 176$$

Hence, point (3.4,1.2) in world = point (332,176) on screen

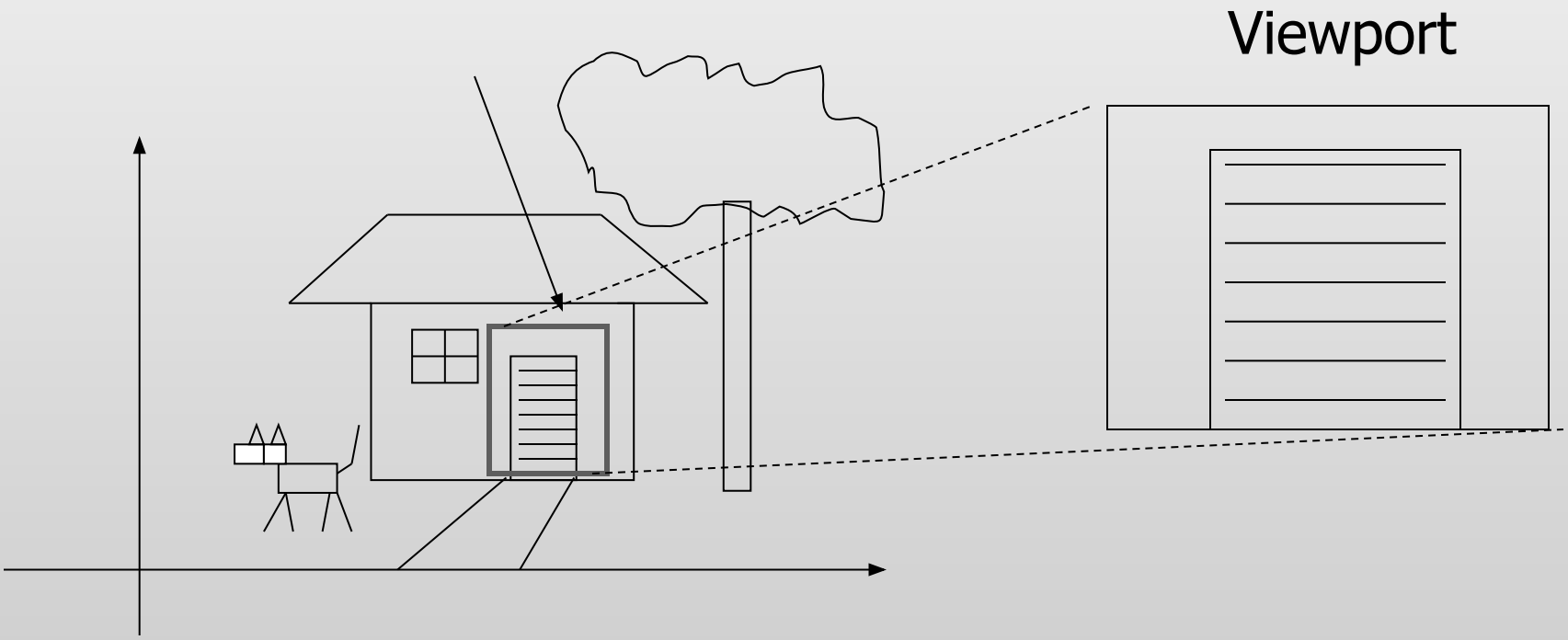
World window set up

- Find the world coordinates extent that will cover the entire scene



Zoom into the picture

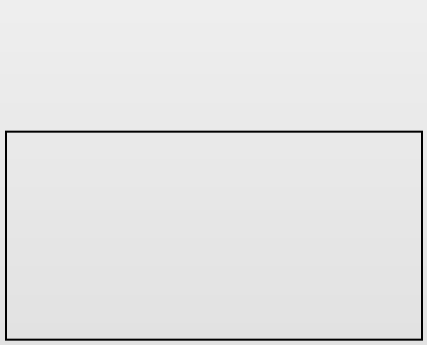
Shrink your world window – call gluOrtho2D() with a new range



Non-distorted viewport setup

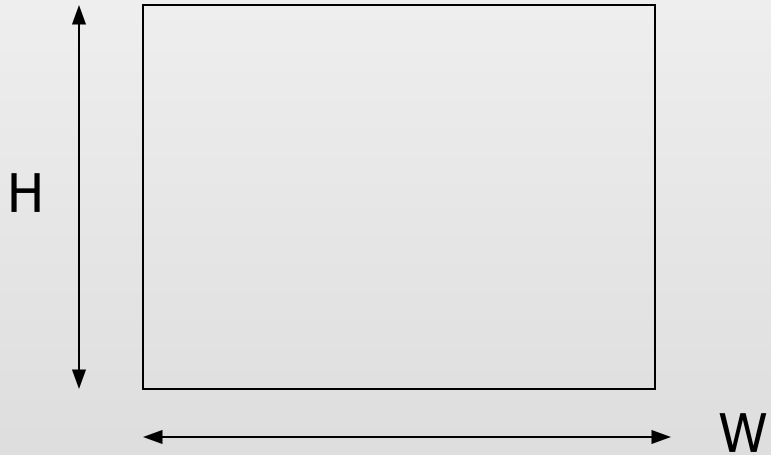
- Distortion happens when …
- World window and display window have different aspect ratios
- Aspect ratio is R = W / H
  - **CASE 1: R > W / H**
  - **CASE 2: R < W / H**

Compare aspect ratios



World window

Aspect Ratio = R



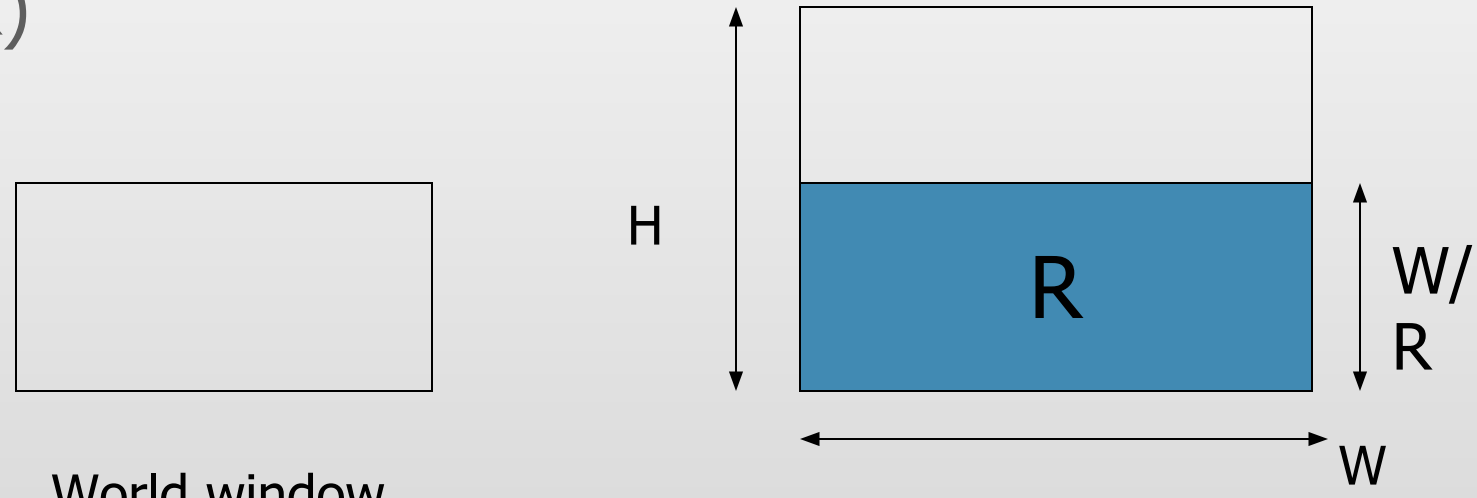
Display window

Aspect Ratio = W / H

R > W / H

CASE 1: R > W / H

glViewport(0, 0, W, W/ R)



World window

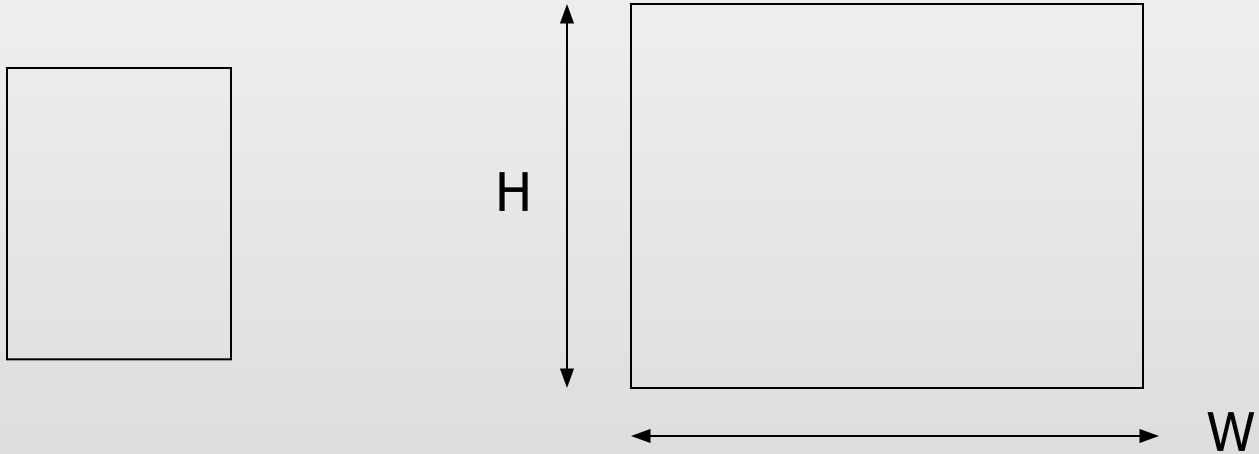
Aspect Ratio = R

Display window

Aspect Ratio = W / H

R > W / H

Case 2: R < W / H



World window

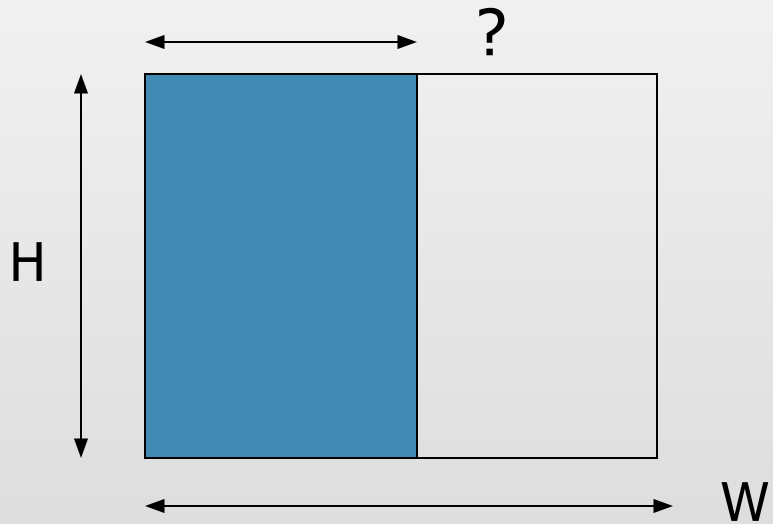
Aspect Ratio = R

Display window

Aspect Ratio = W / H

R < W / H

Match Aspect Ratios



World window

Aspect Ratio = R

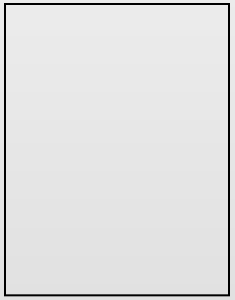
Display window

Aspect Ratio = W / H

$R < W / H$

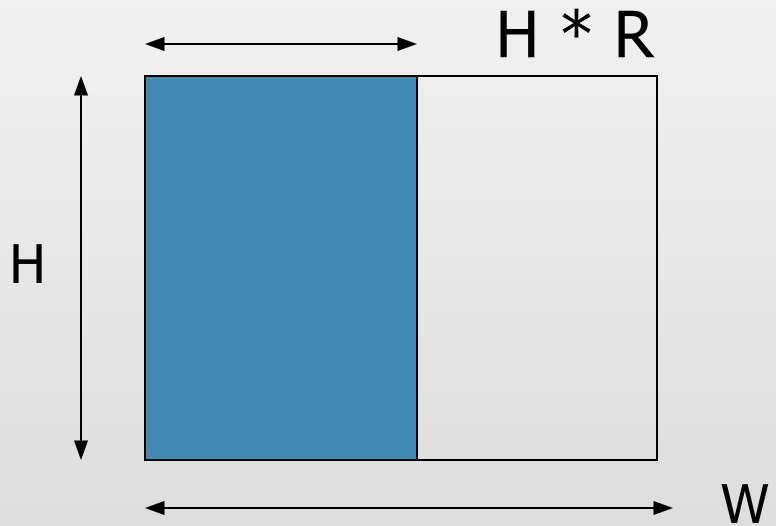
Match aspect ratios

glViewport(0, 0, H*R, H)



World window

Aspect Ratio = R



Display window

Aspect Ratio = W / H

R < W / H

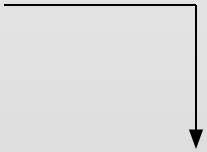
When to call glViewport() ?

Two places:

- Initialization
  - Default: same as the window size
- When the user resizes the display window

Resize (Reshape) window

```
Void main(int argc, char** argv)
{
    ...
    glutDisplayFunc(display);
    glutReshapeFunc(resize);
    glutKeyboardFunc(key);
    ...
}
```



void resize () – a function provided by you. It will be called when the window changes size.

Resize (reshape) window

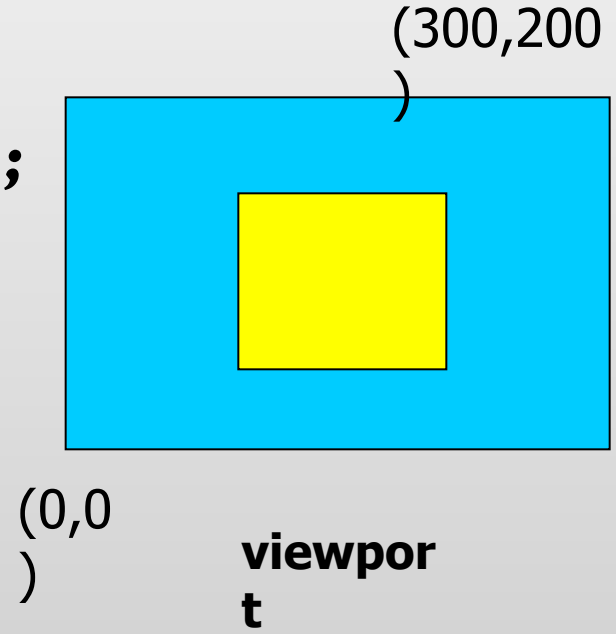
```
Void resize(int W, int H)
{
    glViewport(0,0,W, H);
}
```

This is done by default in GLUT

You can use the call to make sure the aspect ratio is fixed that we just discussed.

Put it all together

```
DrawQuad()
{
  glViewport(0,0,300,200);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-1,1,-1,1);
  glBegin(GL_QUADS);
  glColor3f(1,1,0);
  glVertex2f(-0.5,-0.5);
  glVertex2f(+0.5,-0.5);
  glVertex2f(+0.5,+0.5);
  glVertex2f(-0.5,+0.5);
  glEnd();
}
```



How big is the quad?

Well, this works too …

```
main()
{
…
   glBegin(GL_QUADS);
   glColor3f(1,1,0);
   glVertex2f(-0.5,-0.5);
   glVertex2f(+0.5,0);
   glVertex2f(+0.5,+0.5);
   glVertex2f(-0.5,+0.5);
   glEnd();
}
```

OpenGL Default:

glViewport: as large as you display window

gluOrtho2D:
gluOrtho2D(-1,1,-1,1);

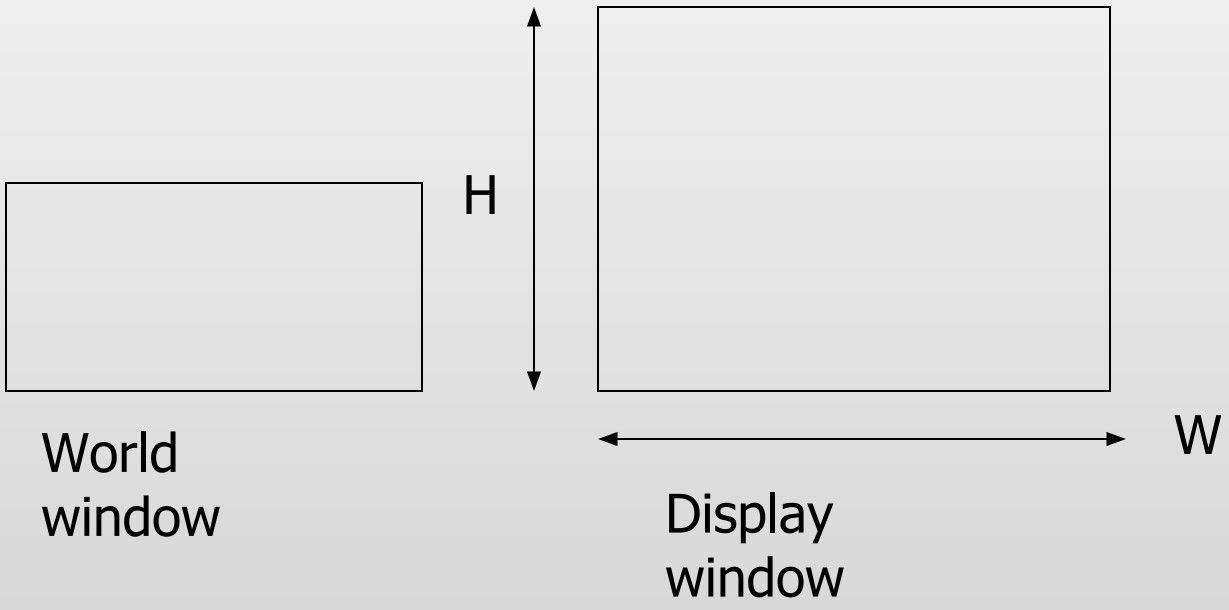
Two viewports

```
void draw(){
// Make background color yellow

glClearColor( 100, 100, 0, 0 );
glClear ( GL_COLOR_BUFFER_BIT );
// Sets up FIRST viewport spanning the left-bottom
    quarter of the interface window
glViewport(0,0,250,250);
// Sets up the PROJECTION matrix
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,50.0,-10.0,40.0); // also sets up world
    window
// Draw BLUE rectangle
glColor3f( 0, 0, 1 );
glRectf(0.0,0.0,10.0,30.0);
// continues
```

```
glViewport(250,250,250,250);
// Sets up the PROJECTION matrix
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,50.0,-10.0,40.0);
// also sets up world window
// Draw RED rectangle
glColor3f( 1, 0, 0 );
glRectf(0.0,0.0,10.0,30.0);
// display rectangles
glutSwapBuffers();
} // end of draw
```

Compare aspect ratios



$R > W / H$

Task3:

What is (Sx,Sy) for point (3.4,1.2) in world coordinates if:

W=(0,4,0,2)

V=(60,380,80,240)

$$W = (W.L, W.R, W.B, W.T) = (0,4,0,2)$$

$$V = (V.L, V.R, V.B, V.T) = (60,380,80,240)$$