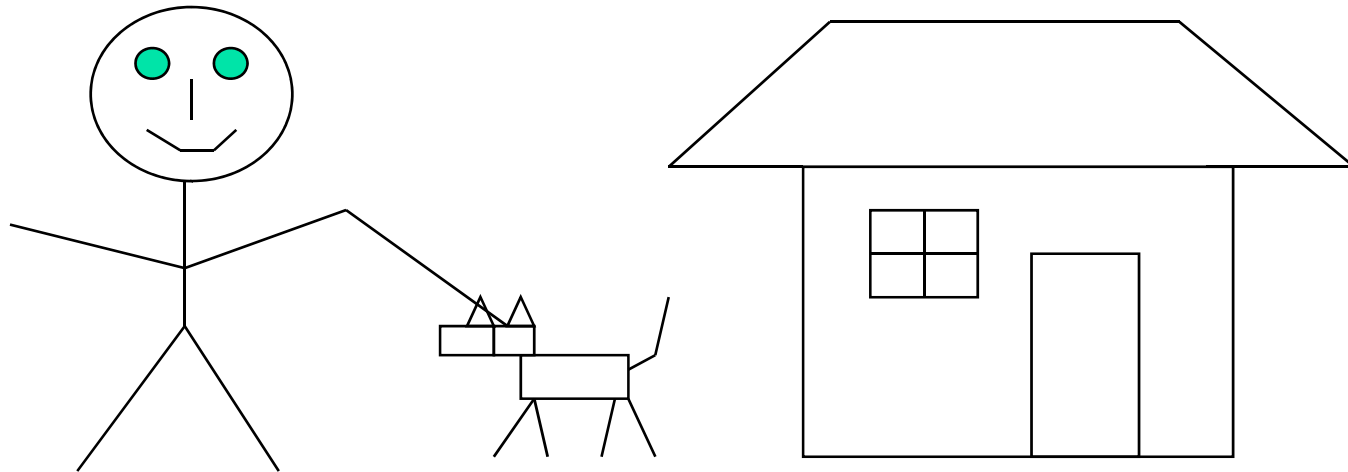




Drawing and Coordinate Systems

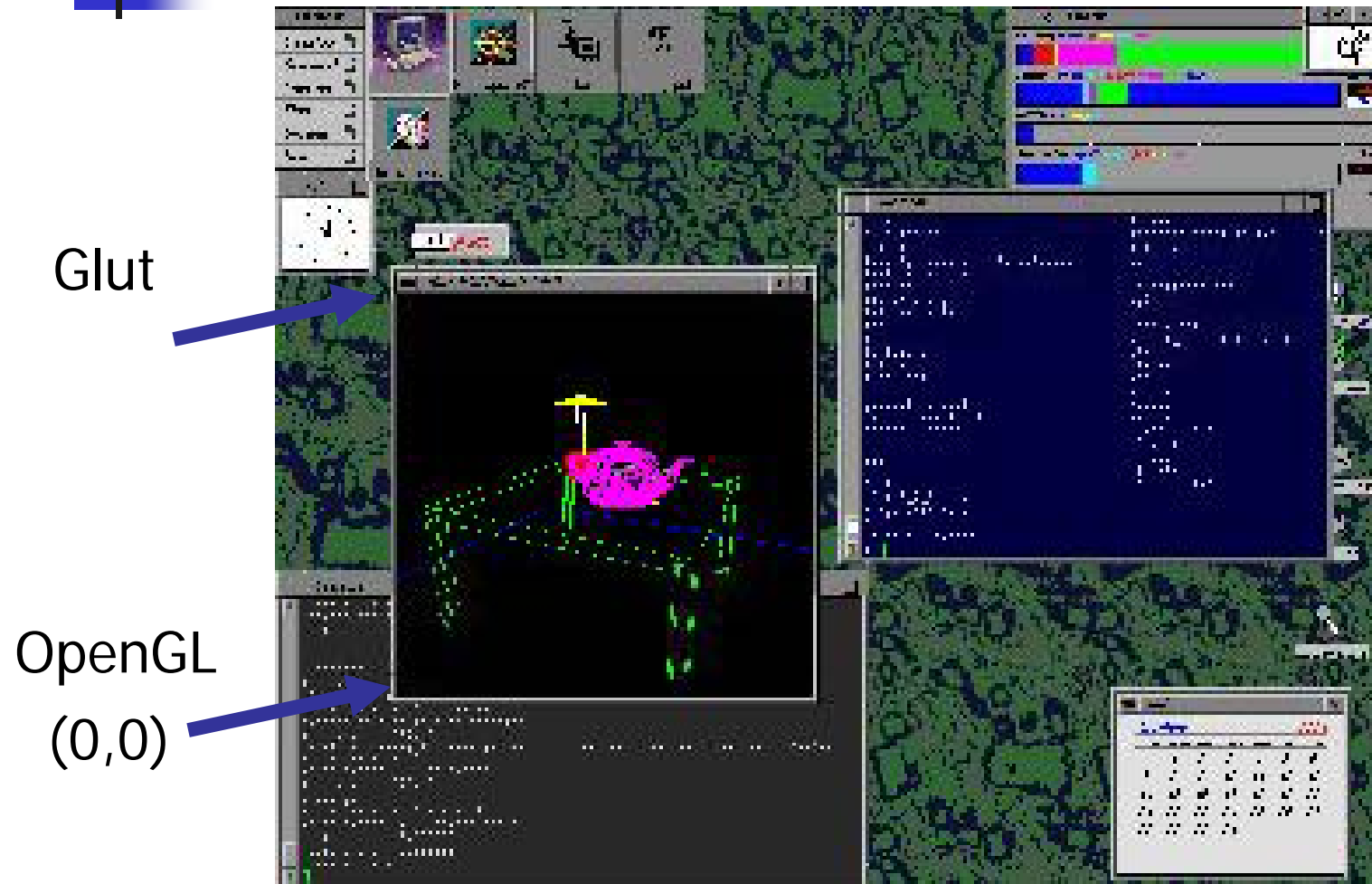




Coordinate Systems

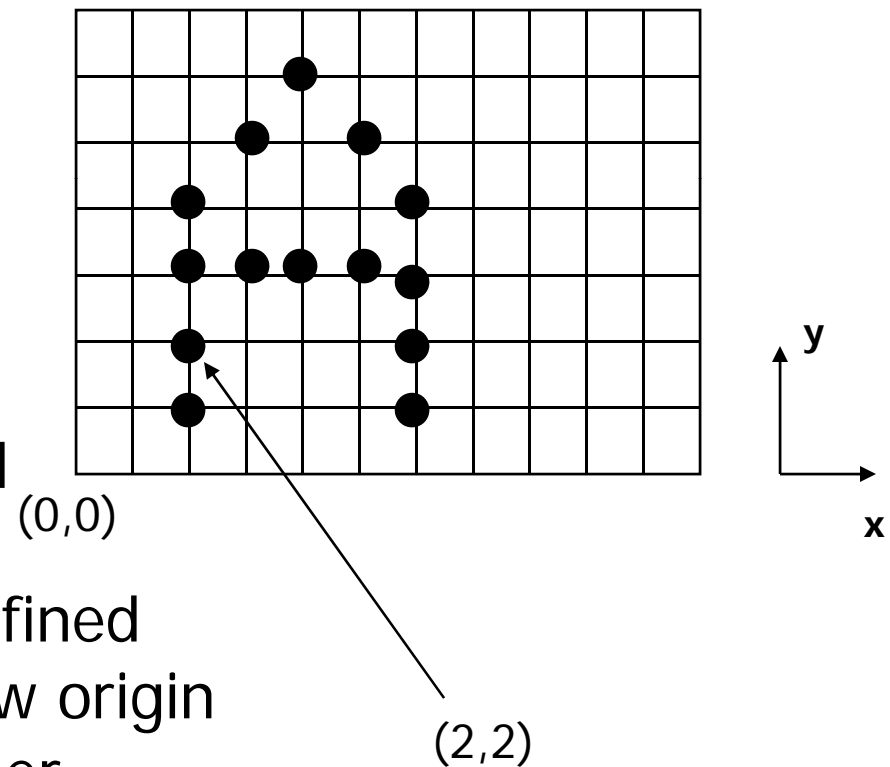
- World Coordinate system
- World window
- Screen Coordinate system
- Viewport
- Window to viewport mapping

Screen Coordinate System



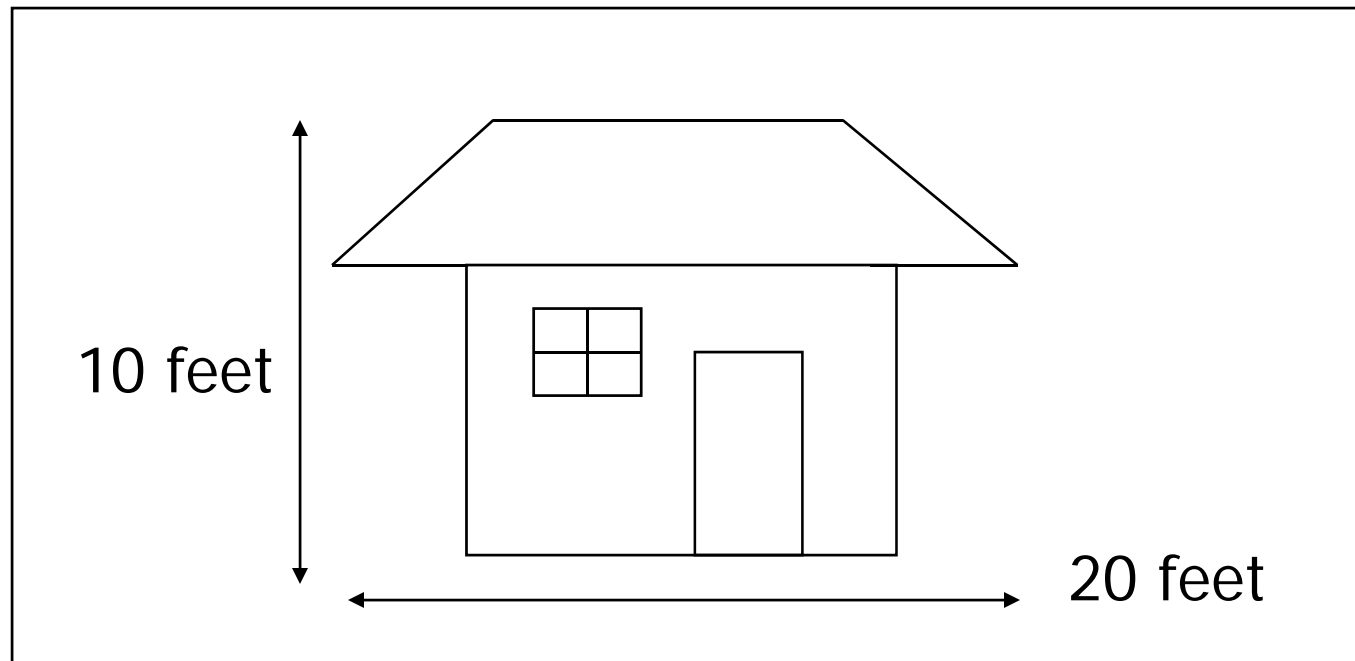
Screen Coordinate System

- 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

- Application specific – difficult to work directly in screen coordinates





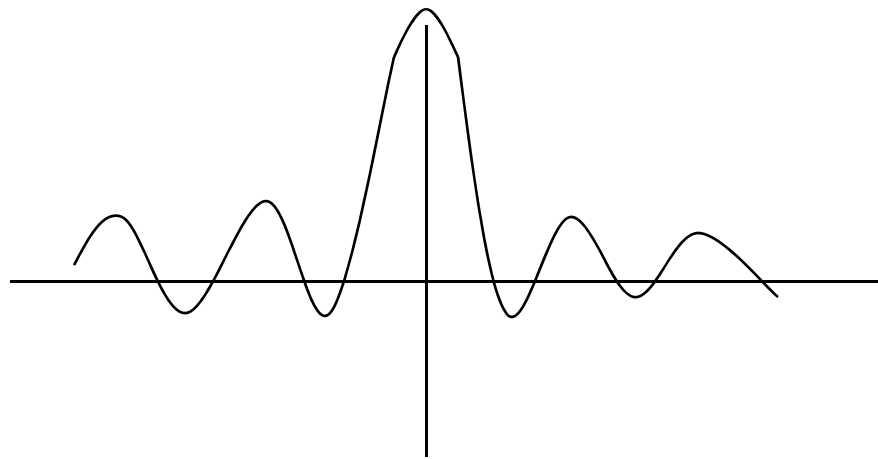
World Coordinate System

- Another example:

plot a sinc function:

$$\text{sinc}(x) = \frac{\sin(\text{PI} * x)}{\text{PI} * x}$$

$$x = -4 \dots +4$$





World Coordinate System

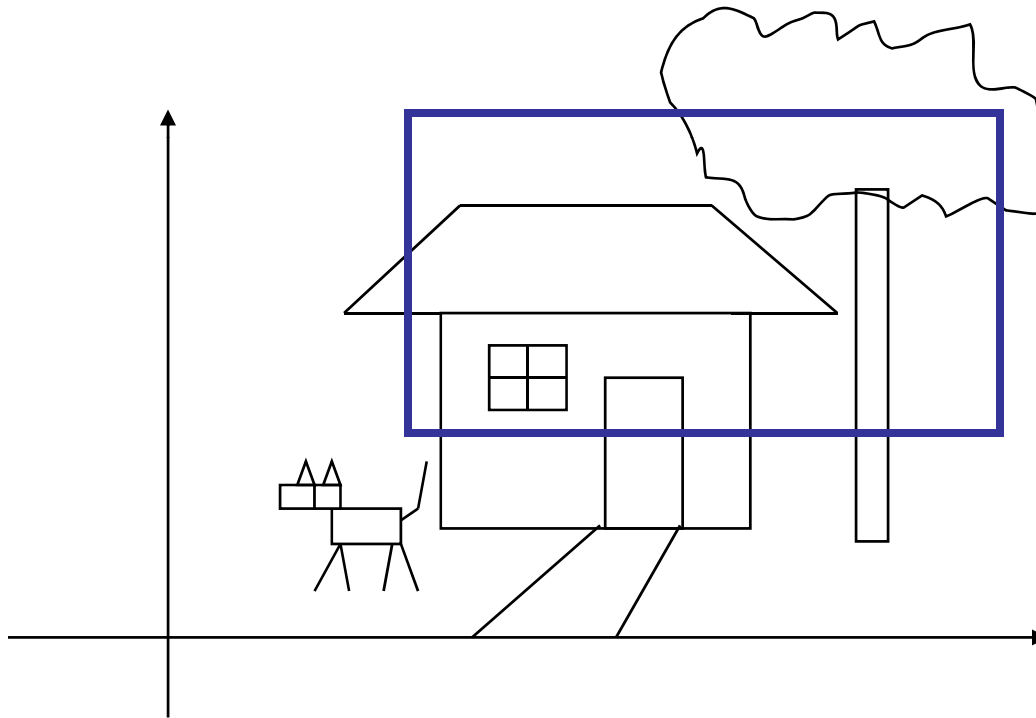
- It would be nice if we can use application specific coordinates – world coordinate system

```
glBegin(GL_LINE_STRIP);
  for (x = -4.0; x <4.0; x+=0.1){
    GLfloat y = sin(3.14 * x) / (3.14 * x);
    glVertex2f (x,y);
  }

glEnd();
```



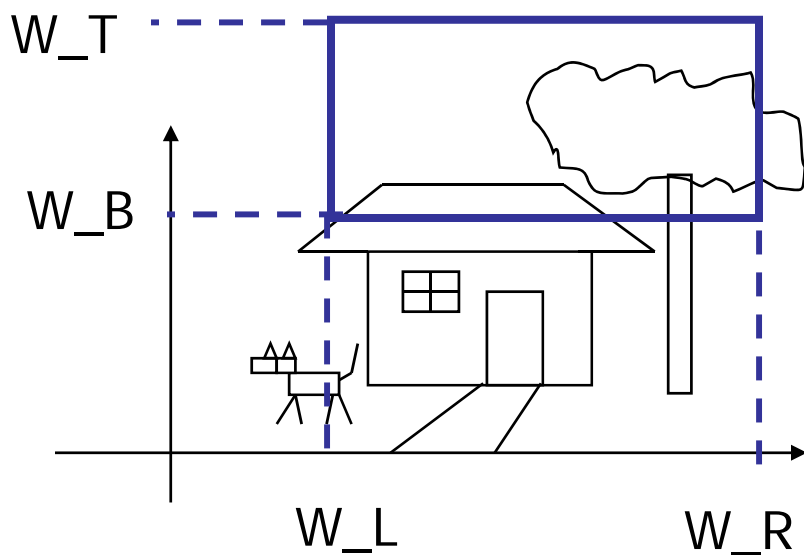
Define a world window





World Window

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



Define by

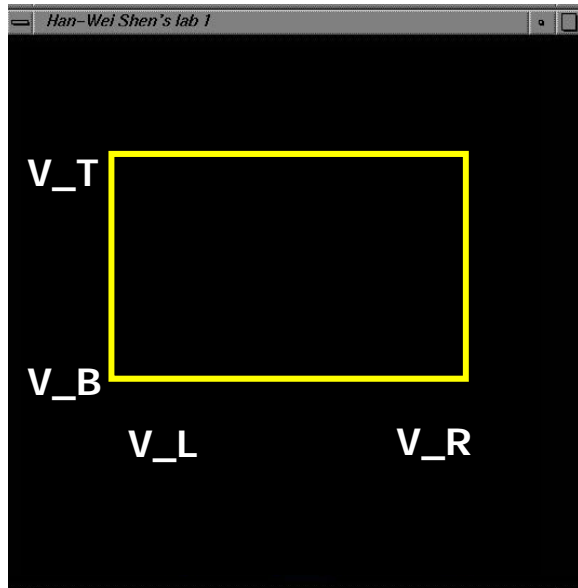
W_L, W_R, W_B, W_T

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())

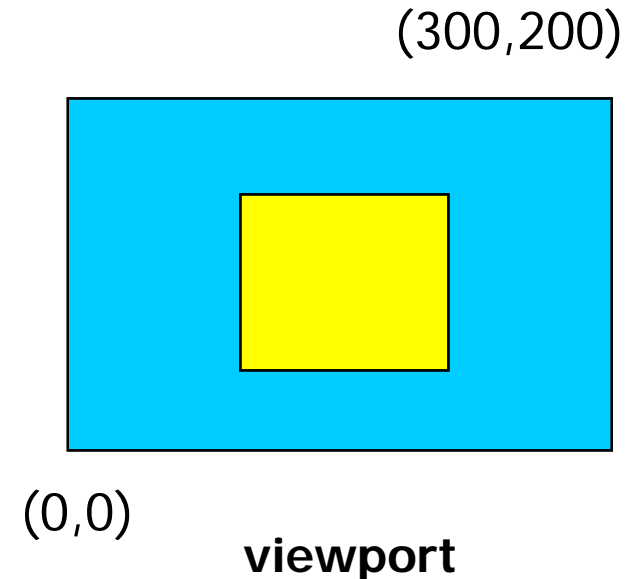


To draw in world coordinate system

- Two 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** (OpenGL internals will do this for you)

A simple example

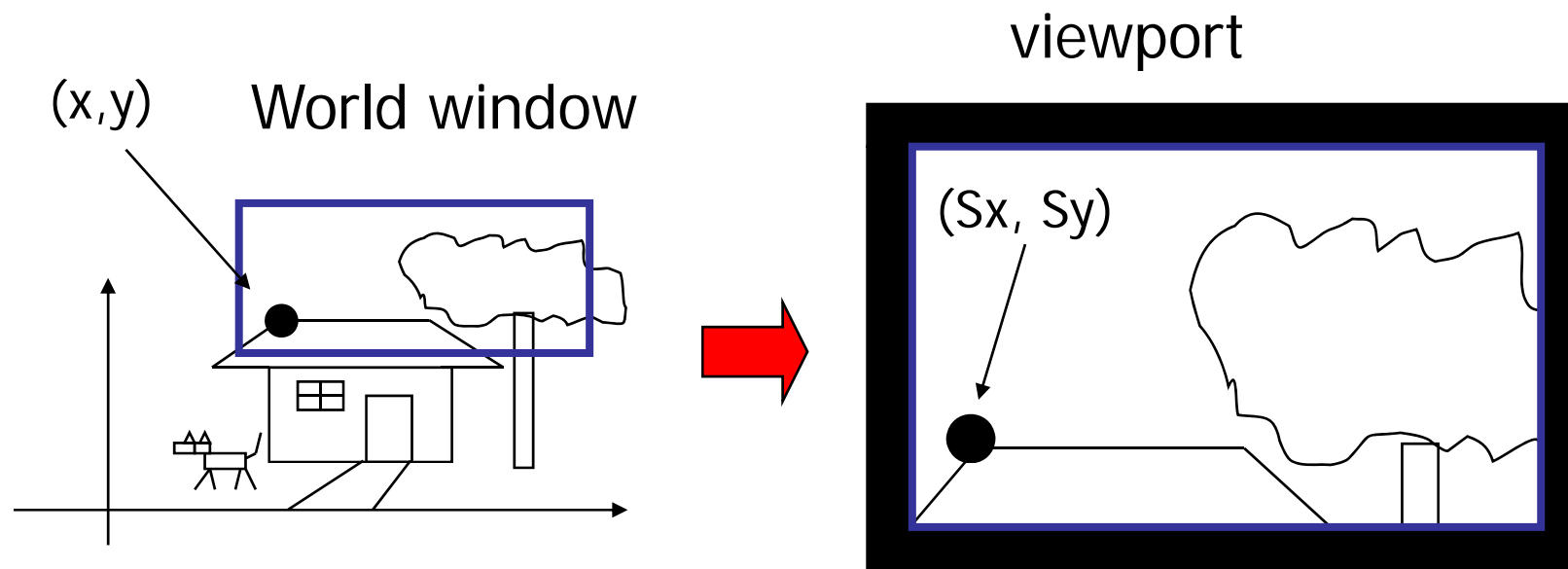
```
DrawQuad()  
{  
    glViewport(0,0,300,200);  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    gluOrtho2D(-1,1,-1,1);  
    glBegin(GL_QUADS);  
    glColor3f(1,1,0);  
    glVertex2f(-0.5,-0.5);  
    glVertex2f(+0.5,-0.5);  
    glVertex2f(+0.5,+0.5);  
    glVertex2f(-0.5,+0.5);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



How big is the quad?

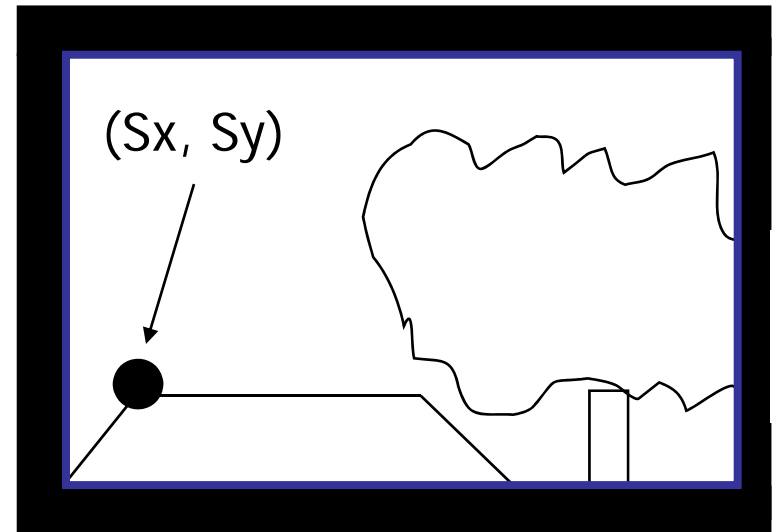
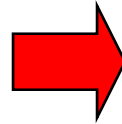
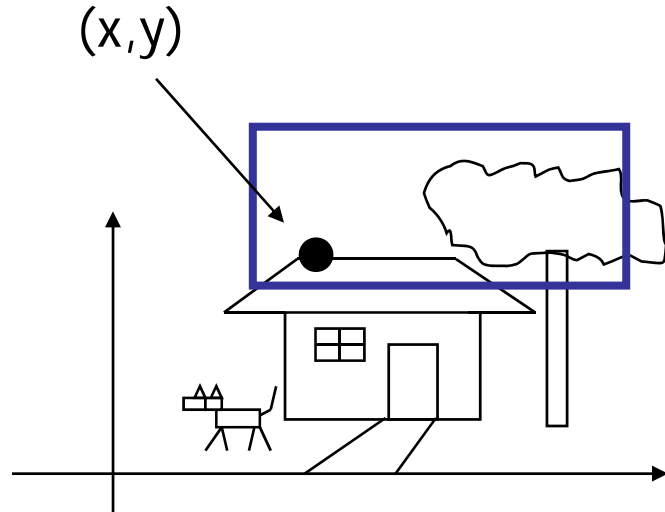
Window to viewport mapping

- 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

- First thing to remember – you don't need to do it by yourself. OpenGL will do it for you
 - You just need to define the viewport (with `glViewport()`), and the world window (with `gluOrtho2D()`)
- But we will look 'under the hood'



Also, one thing to remember ...

- A practical OpenGL issue
 - Before calling `gluOrtho2D()`, you need to have the following two lines of code –

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
gluOrtho2D(Left, Right, Bottom, Top);
```

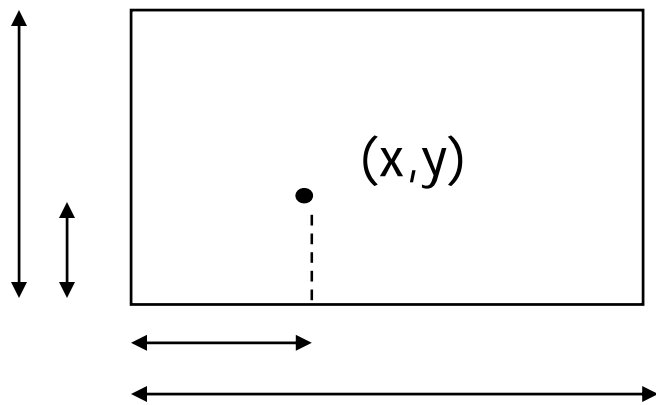


Window to viewport mapping

- Things that are given:
 - The world window (W_L, W_R, W_B, W_T)
 - The viewport (V_L, V_R, V_B, V_T)
 - A point (x, y) in the world coordinate system
- Calculate the corresponding point (s_x, s_y) in the screen coordinate system

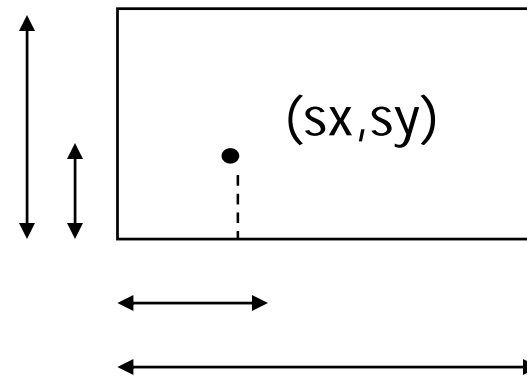
Window to viewport mapping

- Basic principle: the mapping should be proportional



$$(x - W_L) / (W_R - W_L)$$

$$(y - W_B) / (W_T - W_B)$$



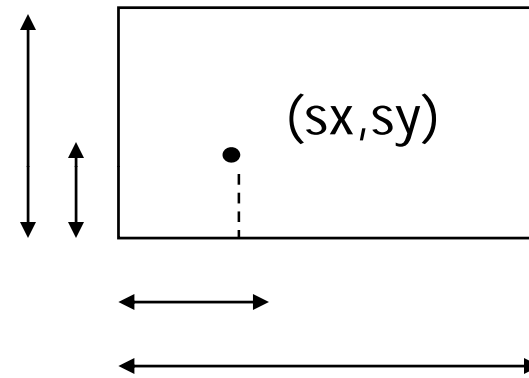
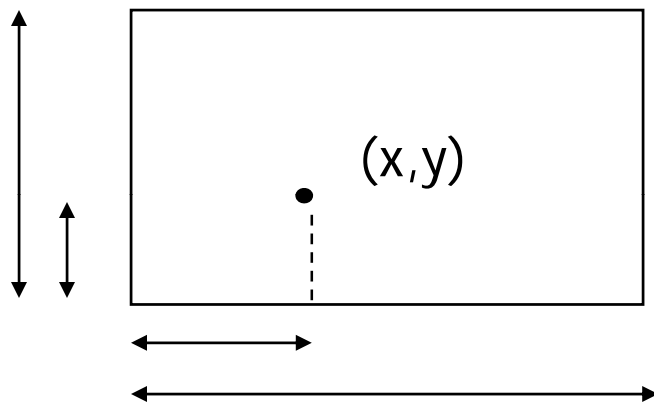
$$(s_x - V_L) / (V_R - V_L)$$

$$(s_y - V_B) / (V_T - V_B)$$

=

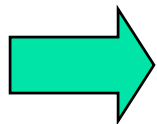
=

Window to viewport mapping



$$(x - W_L) / (W_R - W_L) = (sx - V_L) / (V_R - V_L)$$

$$(y - W_B) / (W_T - W_B) = (sy - V_B) / (V_T - V_B)$$



$$sx = x * (V_R - V_L) / (W_R - W_L) - W_L * (V_R - V_L) / (W_R - W_L) + V_L$$

$$sy = y * (V_T - V_B) / (W_T - W_B) - W_B * (V_T - V_B) / (W_T - W_B) + V_B$$



Some practical issues

- How to set up an appropriate world window automatically?
- How to zoom in the picture?
- How to set up an appropriate viewport, so that the picture is not going to be distorted?

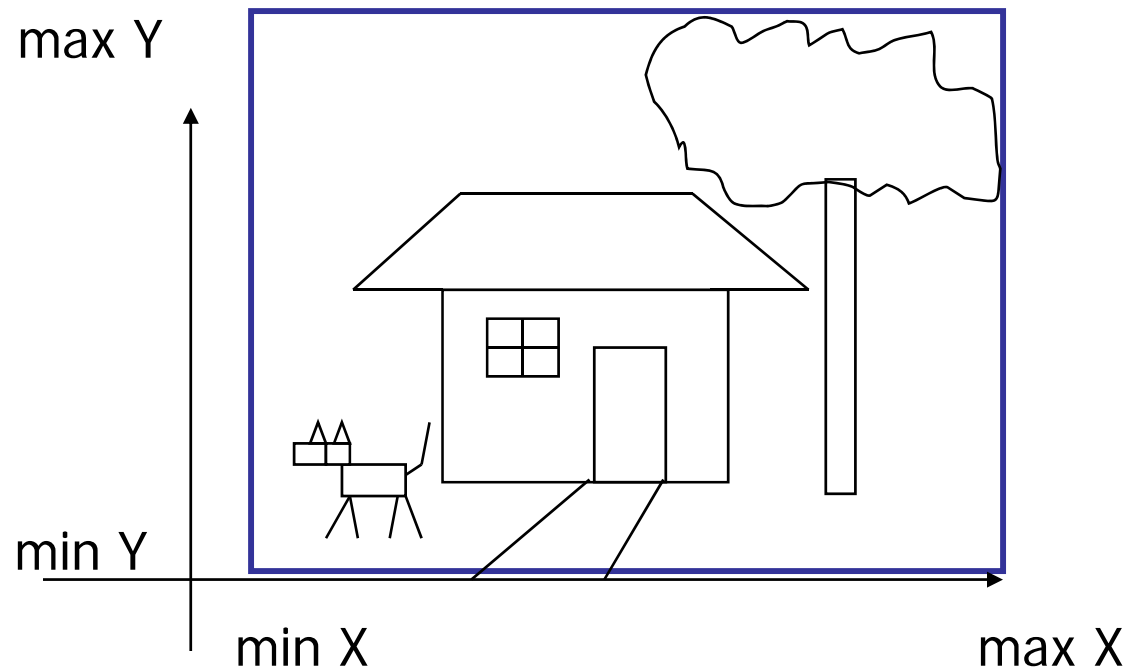


World window setup

- The basic idea is to see all the objects in the world
 - This can just be your initial view, and the user can change it later
- How to achieve it?

World window set up

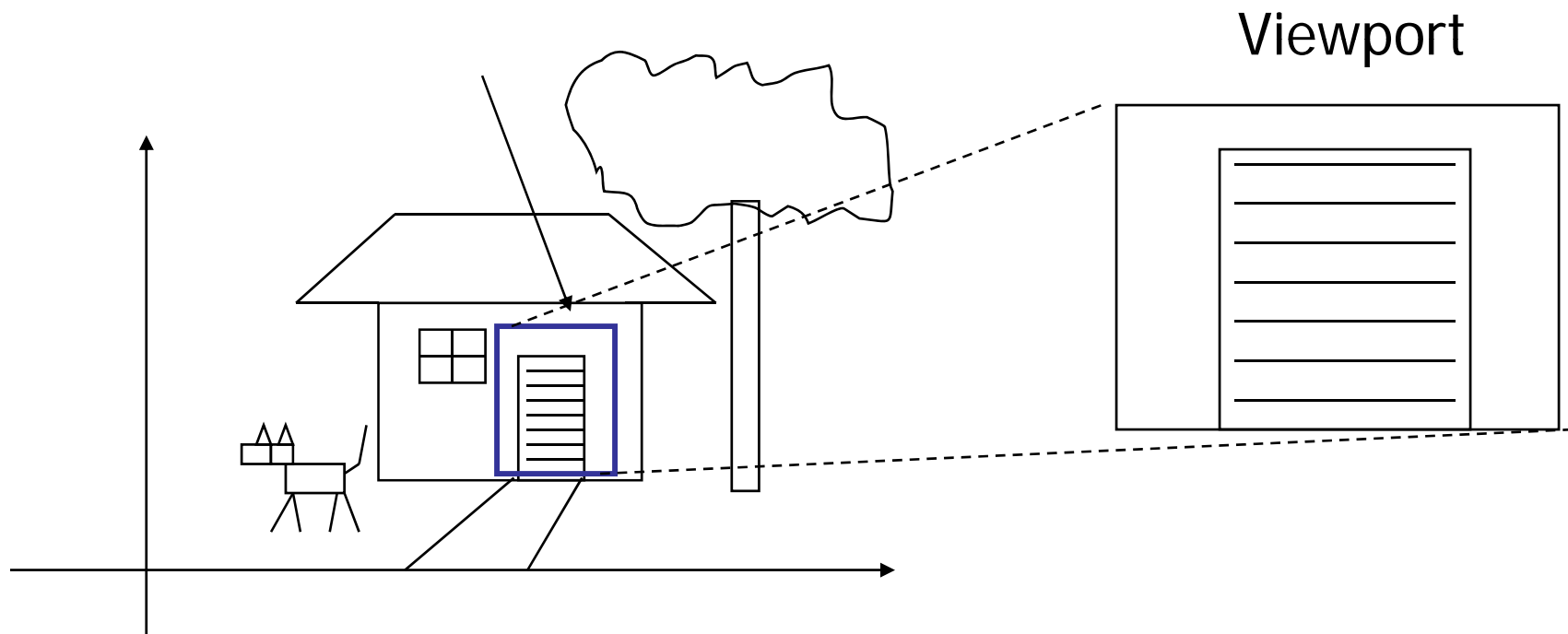
- Find the world coordinates extent that will cover the entire scene





Zoom into the picture

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



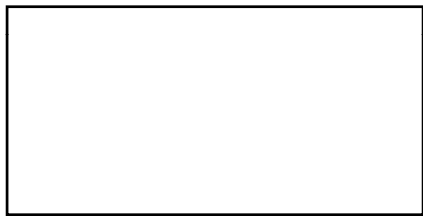


Non-distorted viewport setup

- Distortion happens when ...
- World window and display window have different aspect ratios
- Aspect ratio?
- $R = W / H$

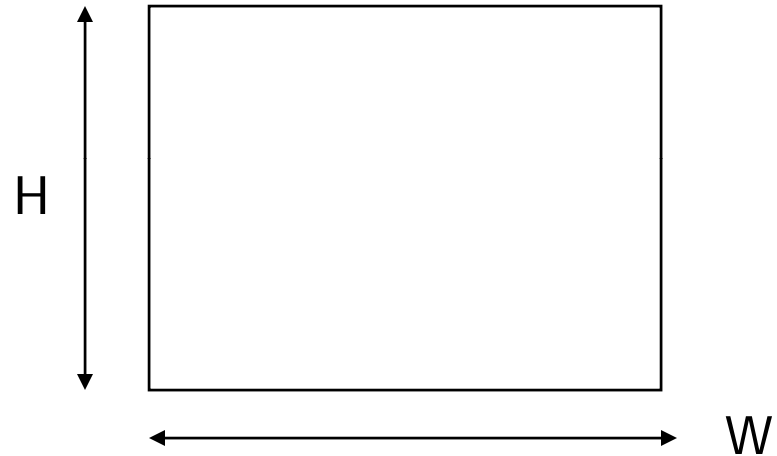


Compare aspect ratios



World window

Aspect Ratio = R



Display window

Aspect Ratio = W / H

$$R > W / H$$

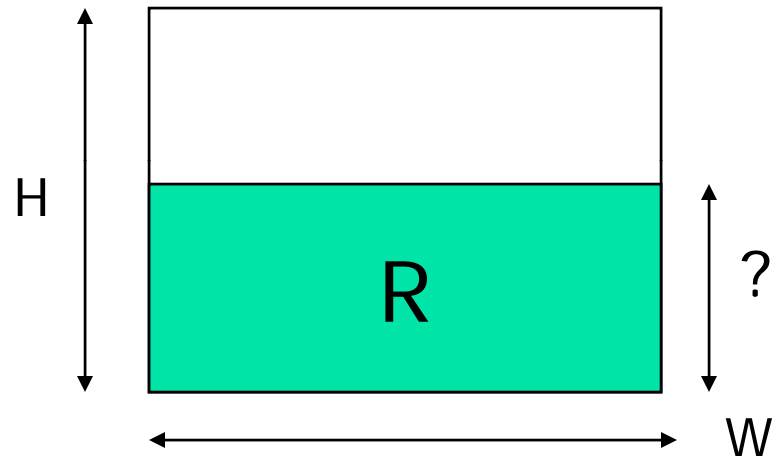


Match aspect ratios



World window

Aspect Ratio = R



Display window

Aspect Ratio = W / H

$$R > W / H$$

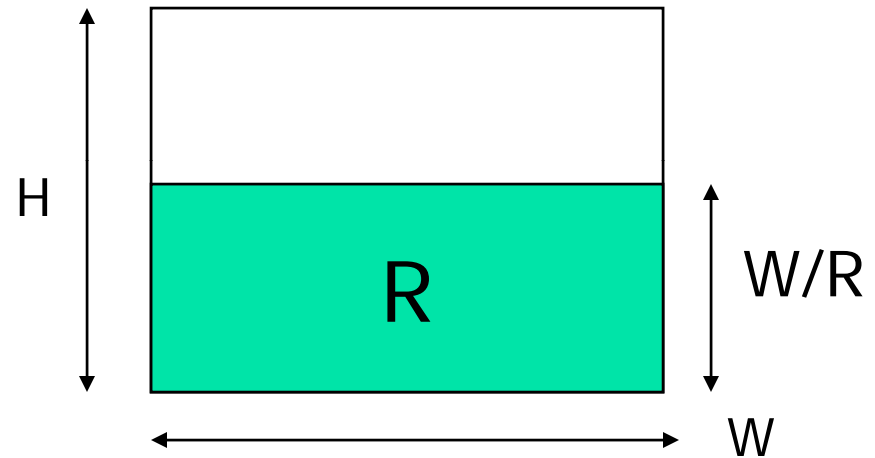
Match aspect ratios



World window

Aspect Ratio = R

$$R > W / H$$



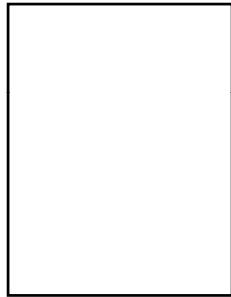
Display window

Aspect Ratio = W / H

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

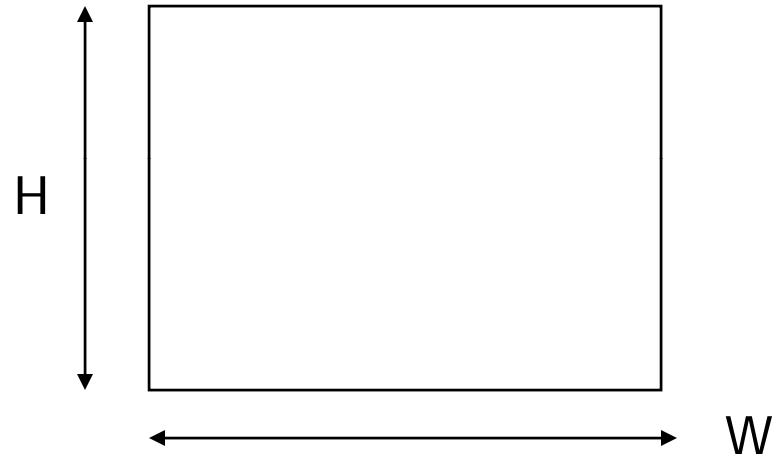


Compare aspect ratios



World window

Aspect Ratio = R



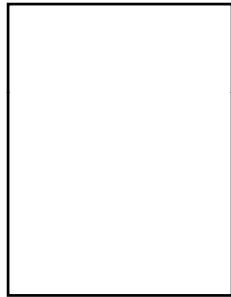
Display window

Aspect Ratio = W / H

$$R < W / H$$

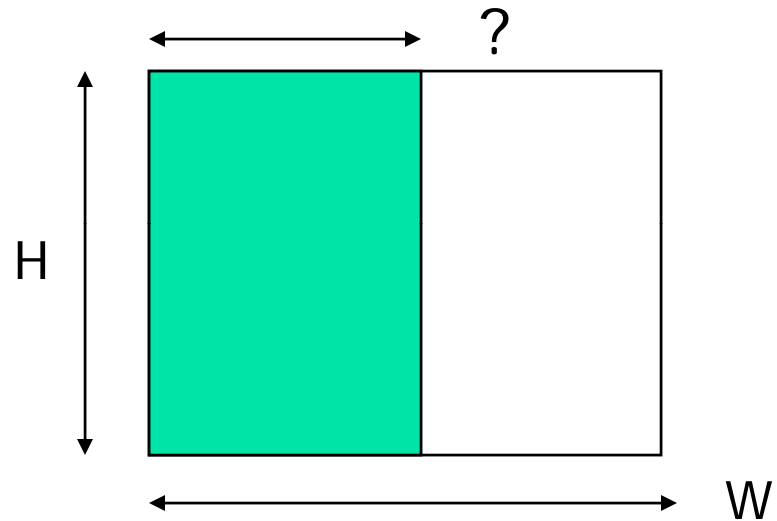


Match aspect ratios



World window

Aspect Ratio = R

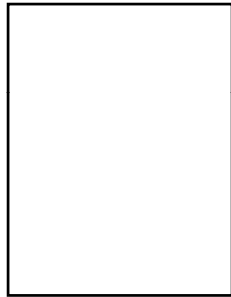


Display window

Aspect Ratio = W / H

$$R < W / H$$

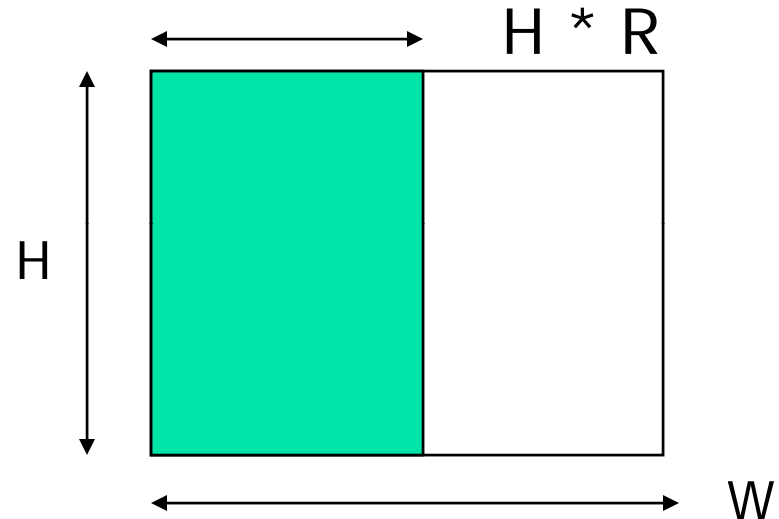
Match aspect ratios



World window

Aspect Ratio = R

$$R < W / H$$



Display window

Aspect Ratio = W / H

`glViewport(0, 0, H * R, 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

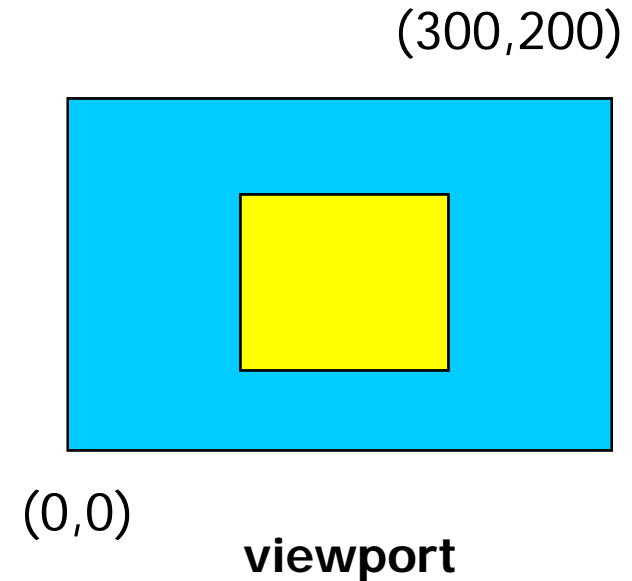
```
Void resize(int W, int H)
{
    glViewport(0,0,W, H);
}
```

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();
}
```

Why?

OpenGL Default:

`glViewport`: as large as
you display window

`gluOrtho2D`:

`gluOrtho2D(-1,1,-1,1);`

Every time you learn a new OpenGL function, always try to know its default arguments