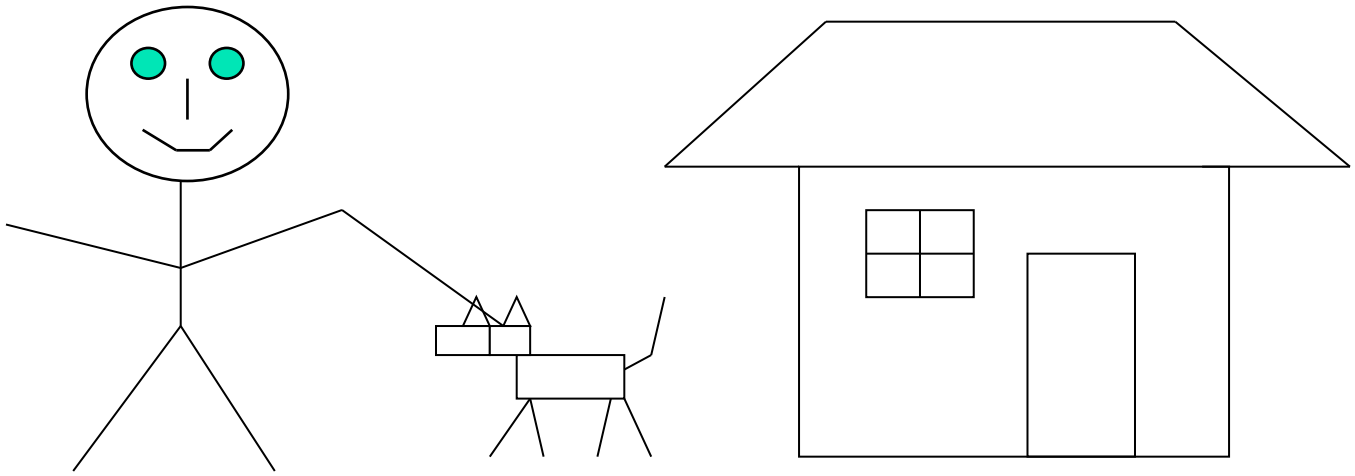
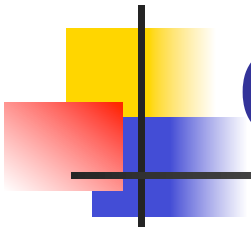
Drawing and Coordinate Systems

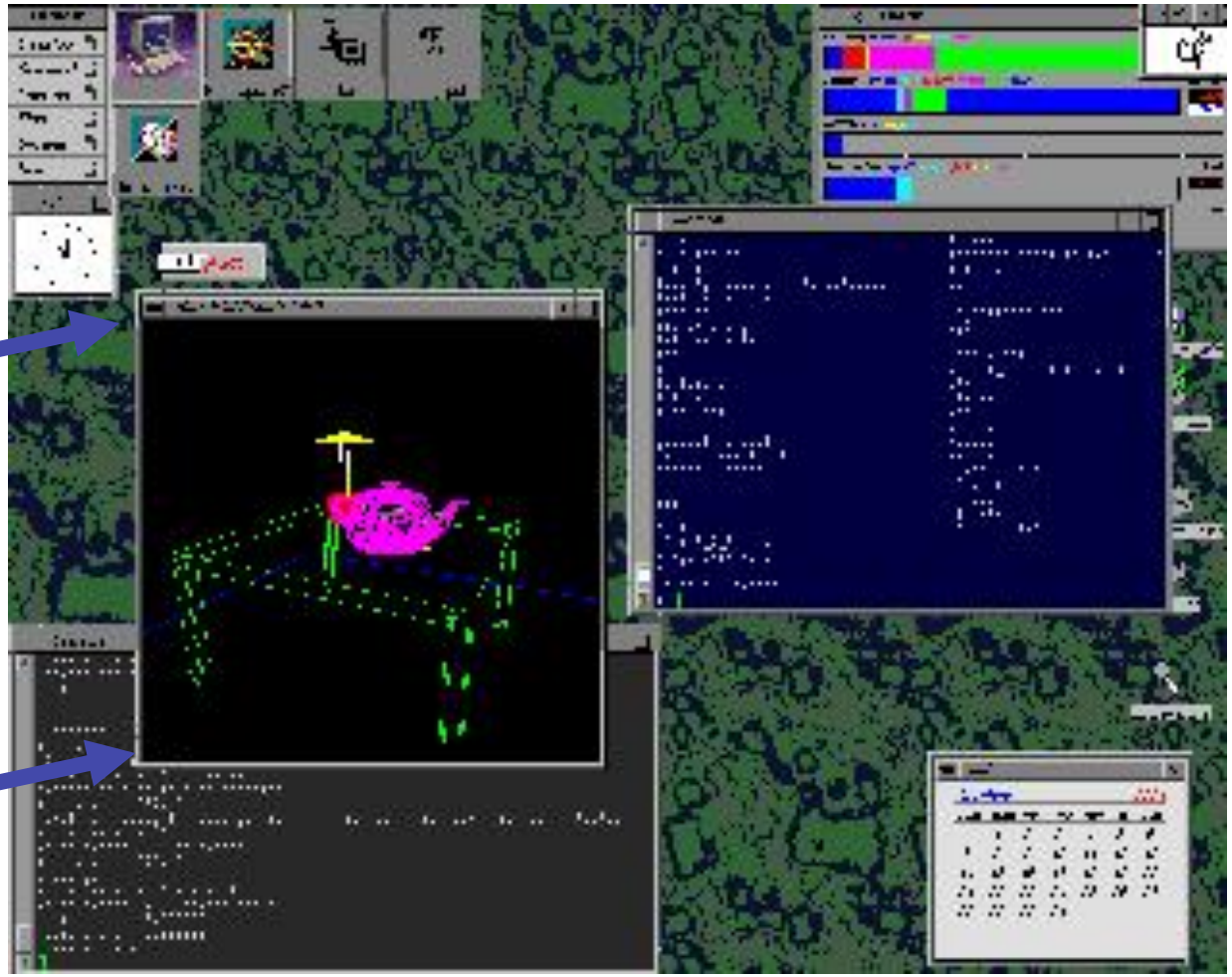




Coordinate Systems

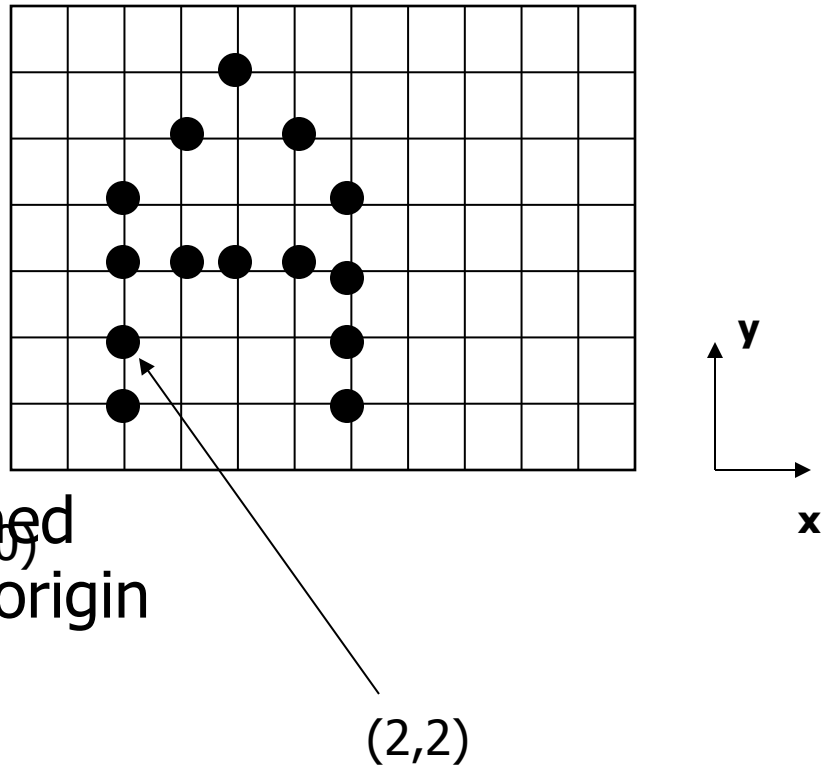
- Screen Coordinate system
- World Coordinate system
- World window
- Viewport
- Window to viewport mapping

Screen Coordinate System



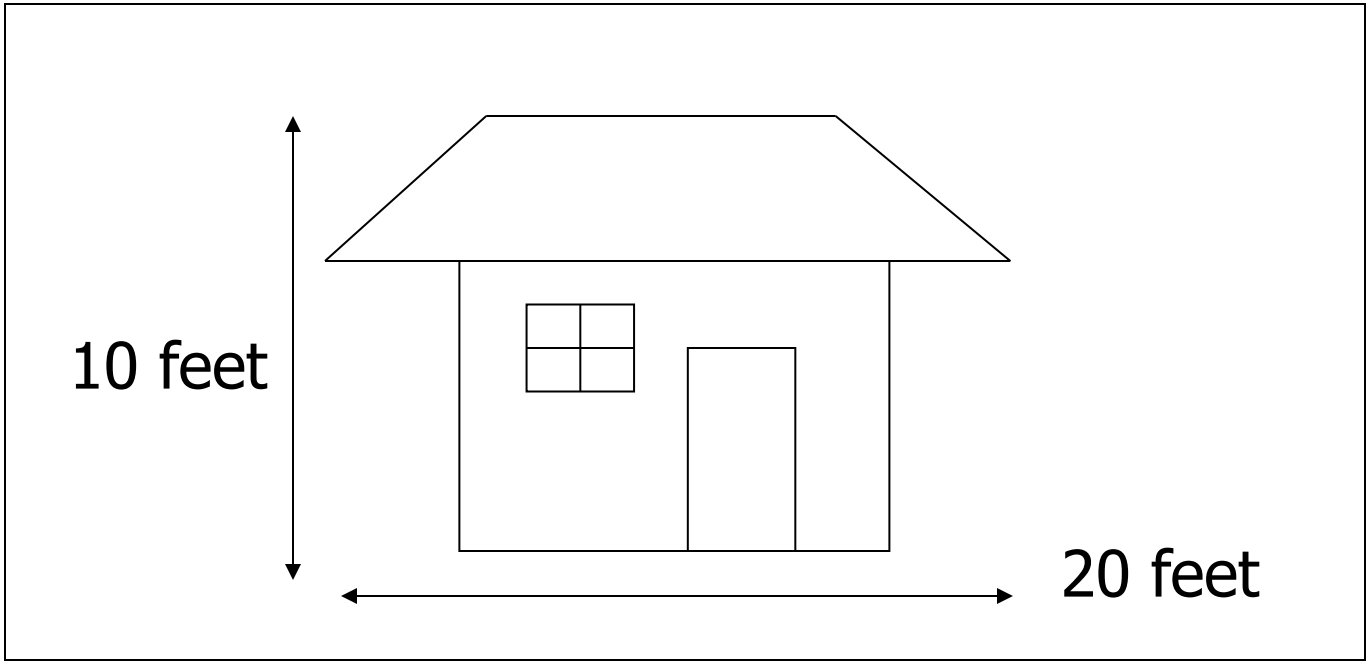
Screen Coordinate System

- 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

- Screen coordinate system is not easy to use



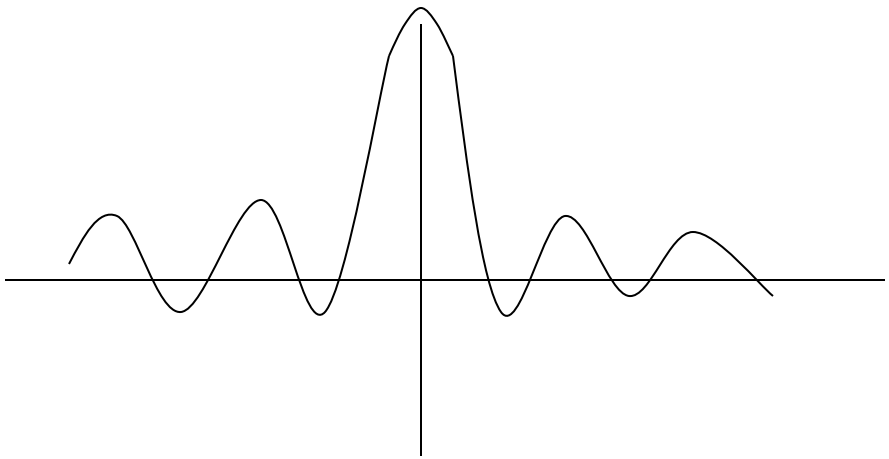
World Coordinate System

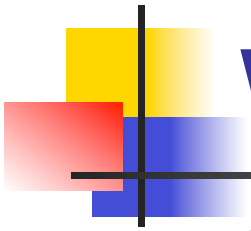
- Another example:

  plot a sinc function:

  sinc(x) = sin(PI*x)/PI*x

  x = -4 .. +4



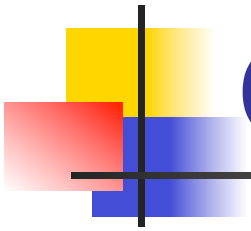


World Coordinate System

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

  world coordinate system

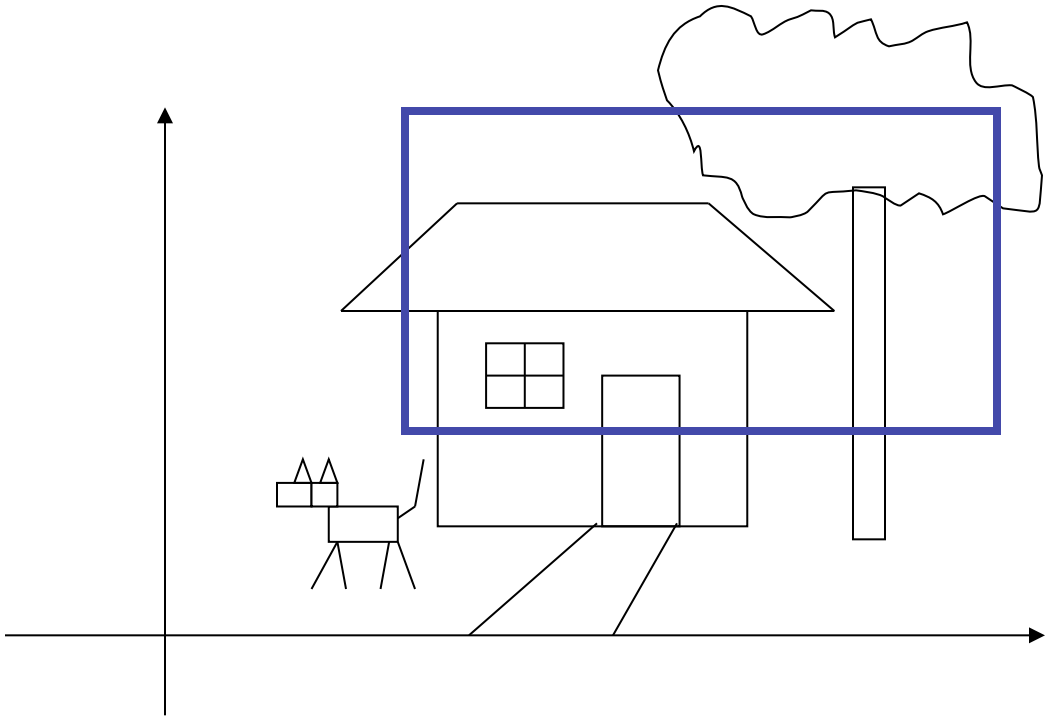
```
glBegin(GL_LINE_STRIP);
  for (x = -4.0; x <4.0; x+=0.1){
    GLfloat y = sin(3.14 * x) / (3.14 * x);
    glVertex2f (x,y);
  }

glEnd();
```



OpenGL 2D Drawing

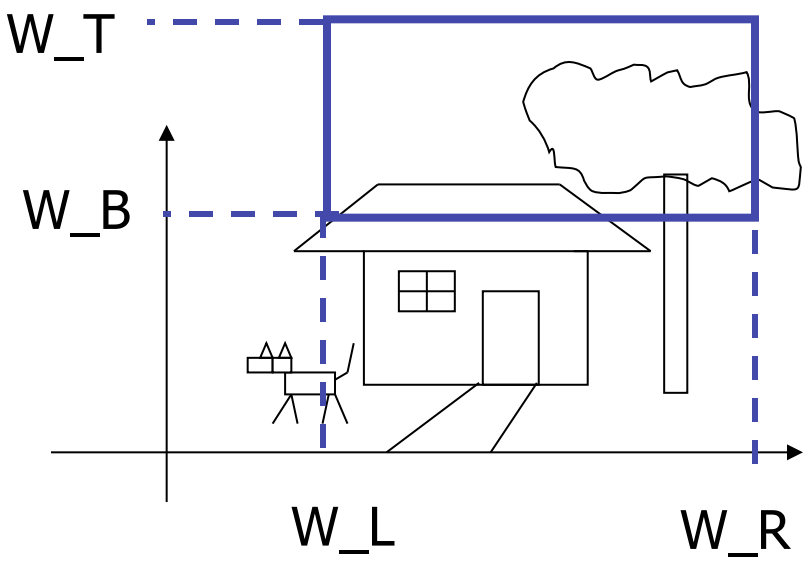
- You can use your own unit/range to define the positions of your objects
- Specify a world window (in world coordinate system) to define what you want to display
- Specify a viewport (in screen coordinate system) to indicate where in the window you want to draw the objects
- OpenGL will do the mapping and drawing for you

Define a world window



World Window

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



Define by

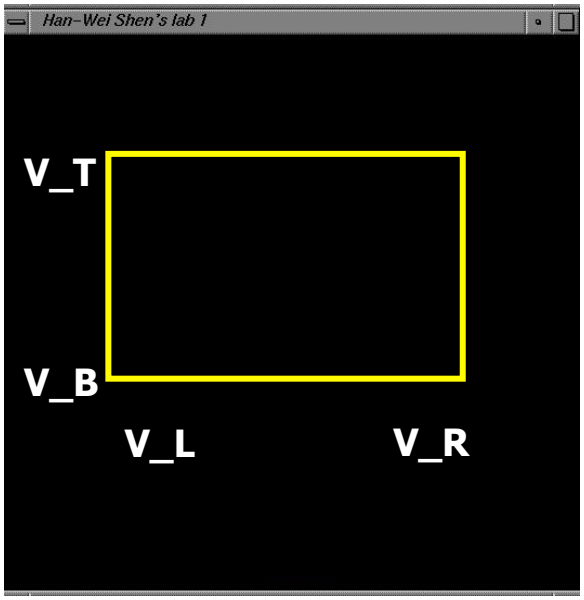
W_L, W_R, W_B, W_T

Use OpenGL command:

gluOrtho2D(left, right, bottom, top)

Viewport

- The rectangular area in the screen used to display the objects contained in the world window
- Defined in the screen coordinate system

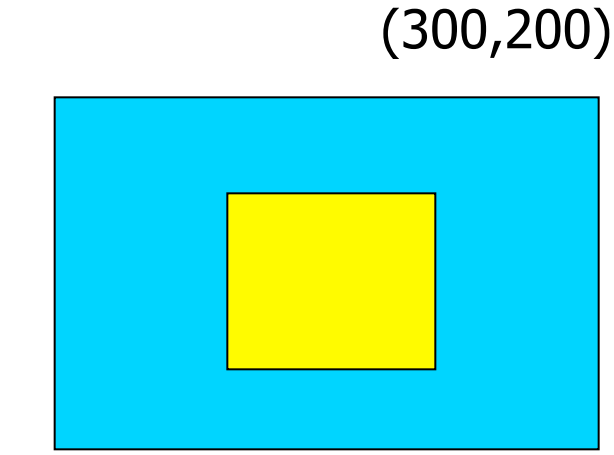



```
glViewport(int  left, int  bottom,
           int  (right-left),
           int (top-bottom));
```

always call this function before drawing

A simple example

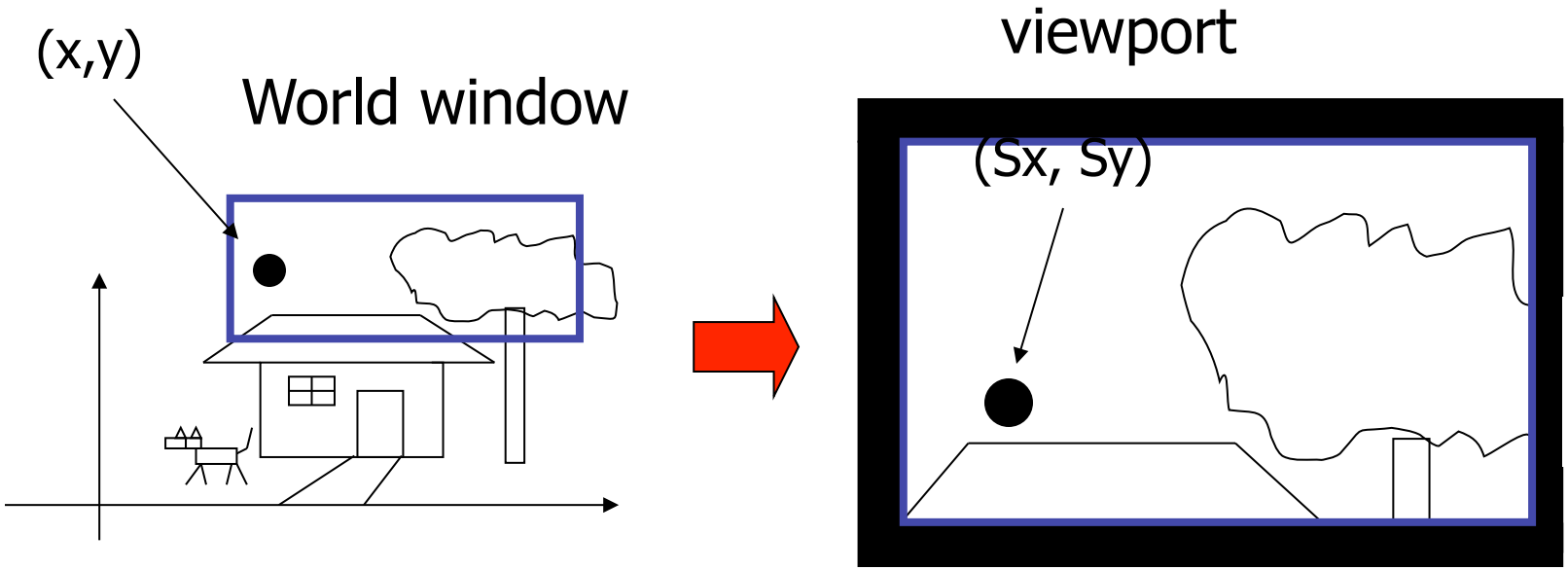
```
DrawQuad()
{
    glViewport(0,0,300,200);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-1,1,-1,1);
    glBegin(GL_QUADS);
    glColor3f(1,1,0);
    glVertex2f(-0.5,-0.5);
    glVertex2f(+0.5,-0.5);
    glVertex2f(+0.5,+0.5);
    glVertex2f(-0.5,+0.5);
    glEnd();
}
```



How big is the quad?

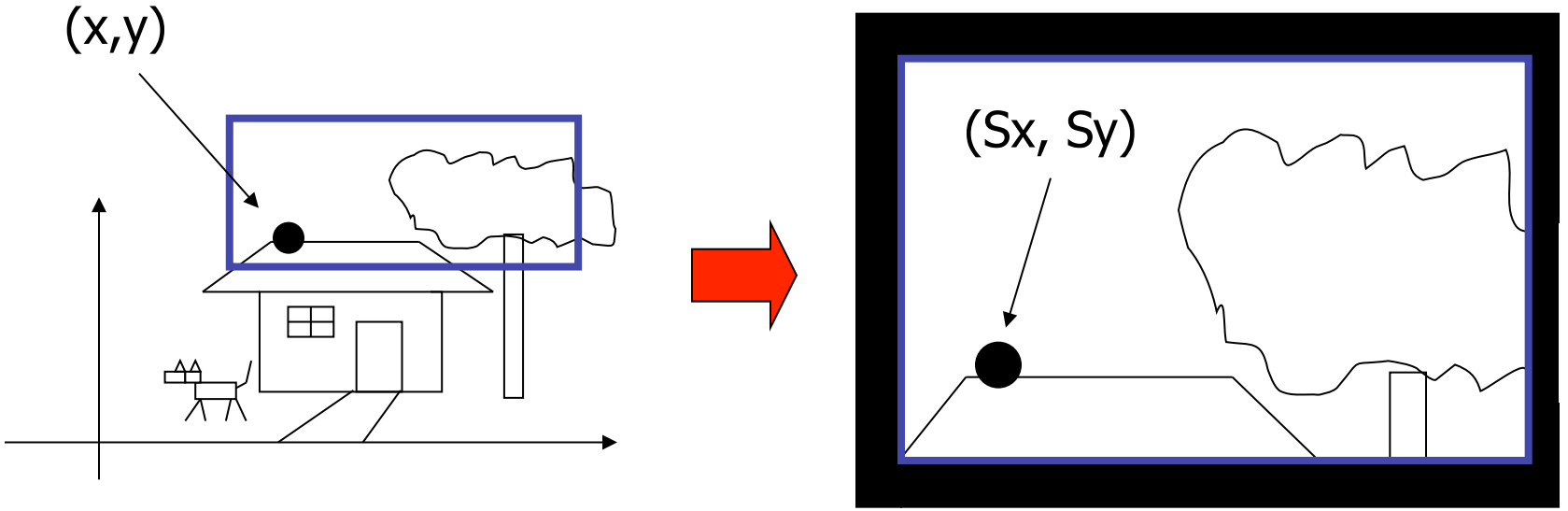
Window to viewport mapping

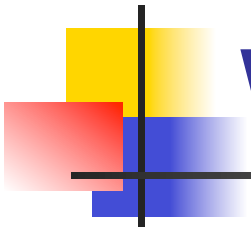
- The objects in the world window will then be drawn onto the viewport



Window to viewport mapping

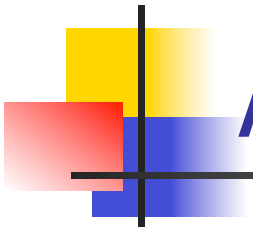
- How to calculate (sx, sy) from (x,y)?





Window to viewport mapping

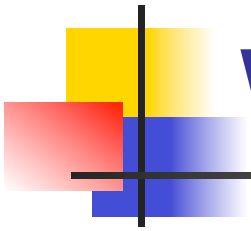
- Remember – you don't need to do it by yourself. OpenGL will do it for you
  - You just need to specify the viewport (with glViewport()), and the world window (with gluOrtho2D())
- But let me explain to you how it is done



Also, one thing to remember …

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

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

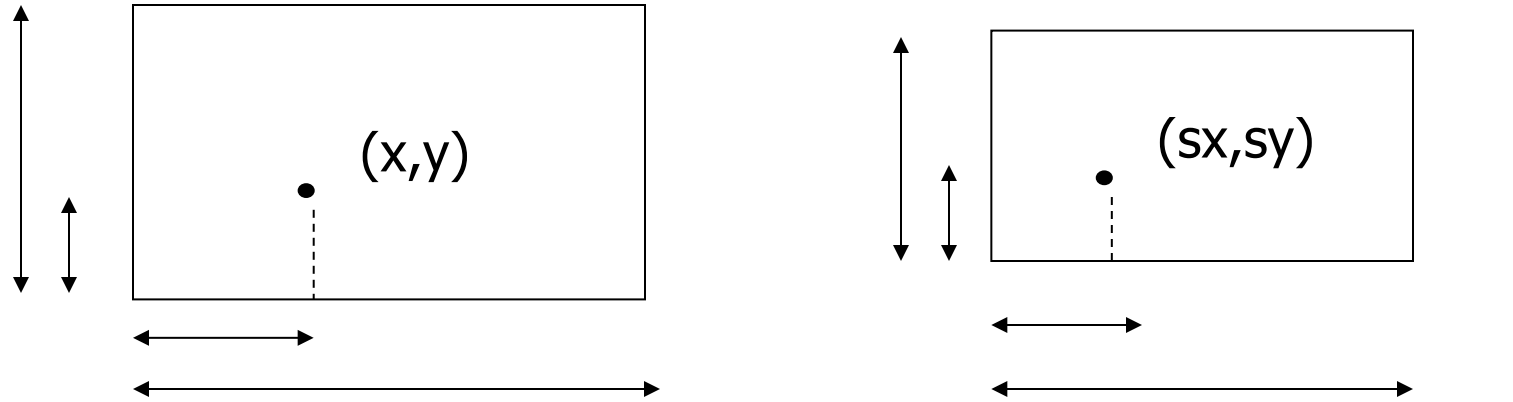


Window to viewport mapping

- 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 (sx, sy) in the screen coordinate system

Window to viewport mapping

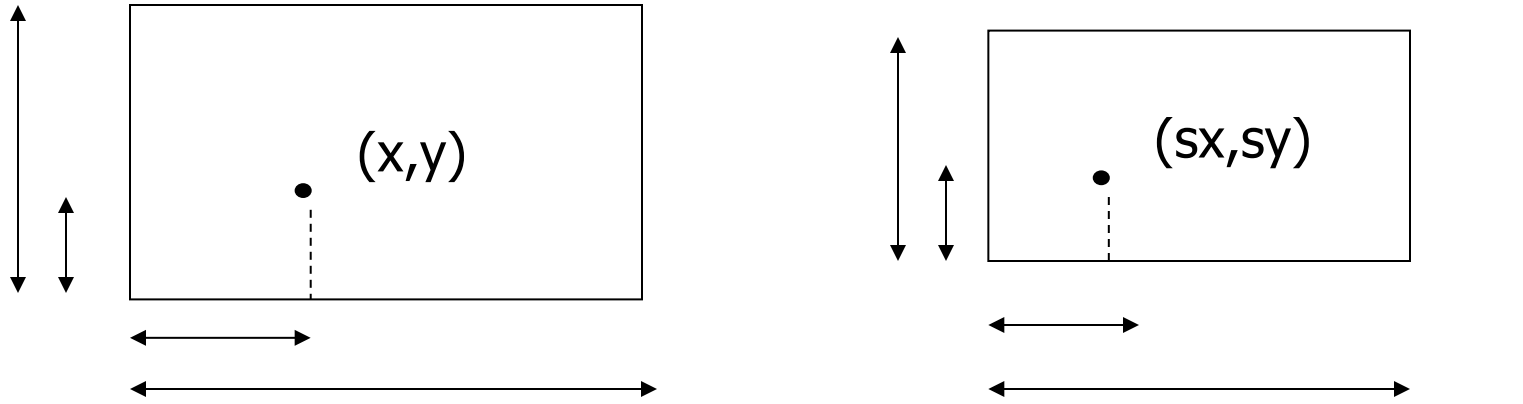
- Basic principle: the mapping should be proportional



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

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

Window to viewport mapping

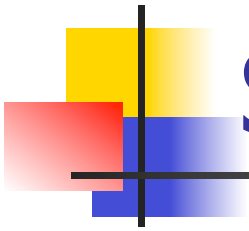


$$(x - W\_L) / (W\_R - W\_L) = (sx - V\_L) / (V\_R - V\_L)$$

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

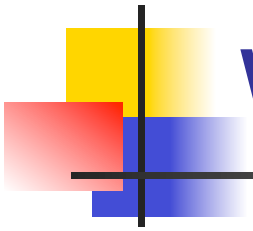
$$sx = x * (V\_R-V\_L)/(W\_R-W\_L) - W\_L * (V\_R - V\_L)/(W\_R-W\_L) + V\_L$$

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



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?

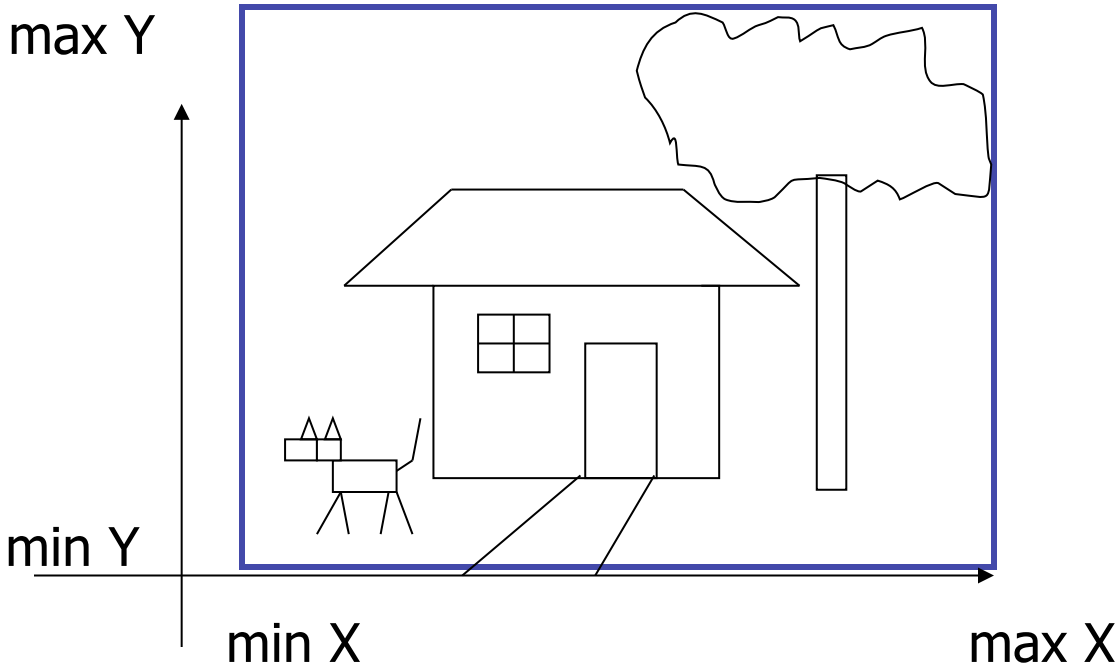


World window setup

- A general approach is to display all the objects in the world
  - This can be your initial view, and the user can change it later
- How to achieve it?

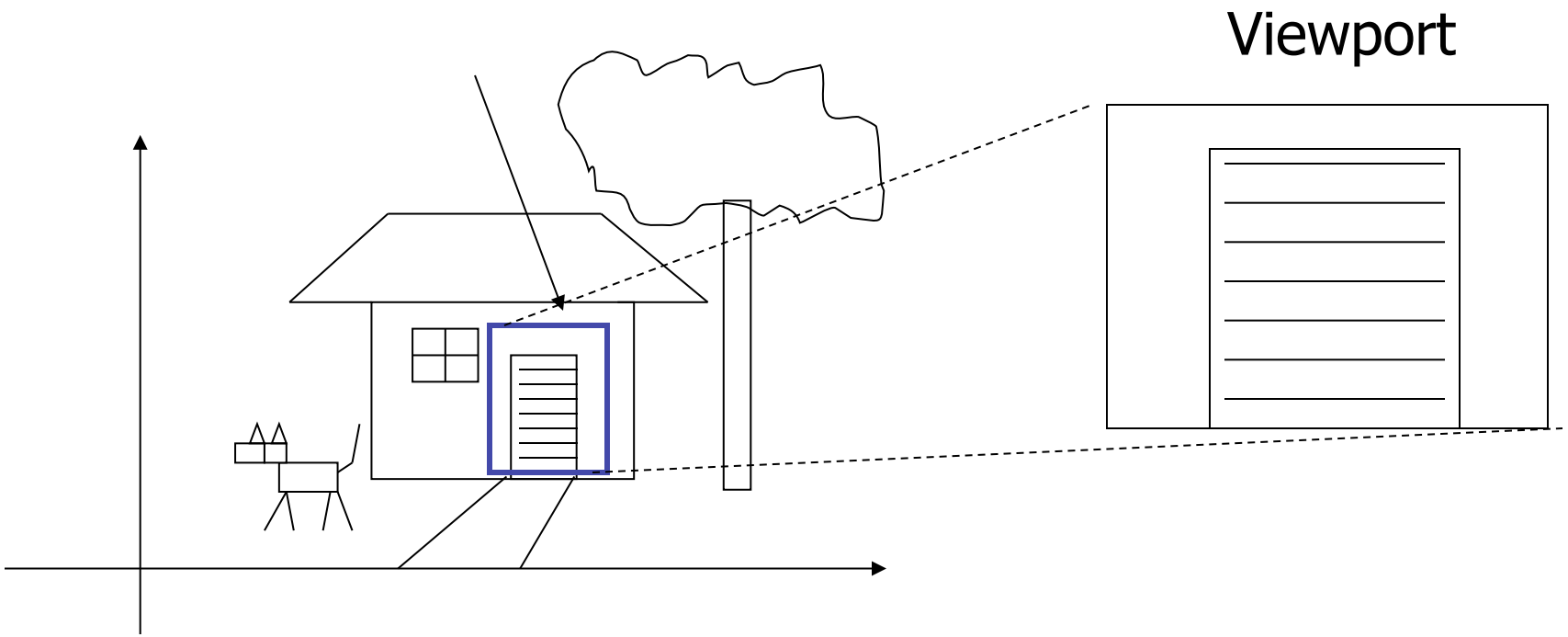
World window set up

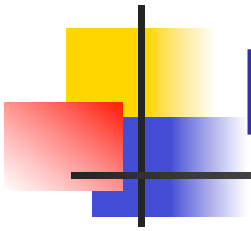
- Find the world coordinates extent that will cover the entire scene



Zoom into the picture

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





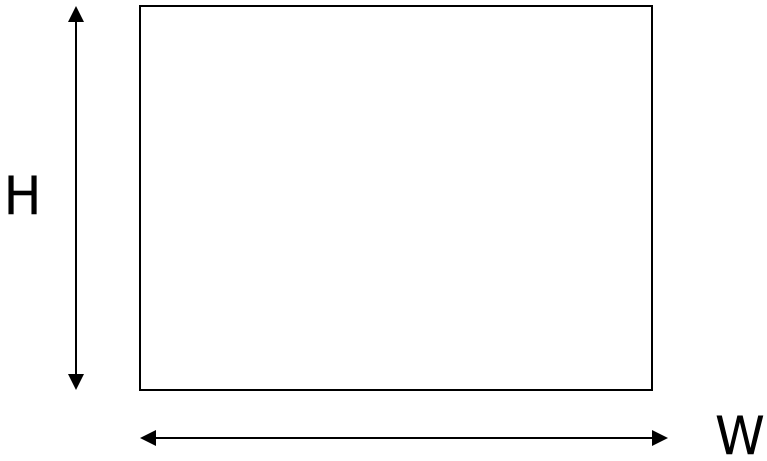
Non-distorted viewport setup

- 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

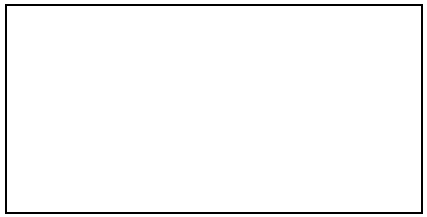


Viewport

Aspect Ratio = W / H

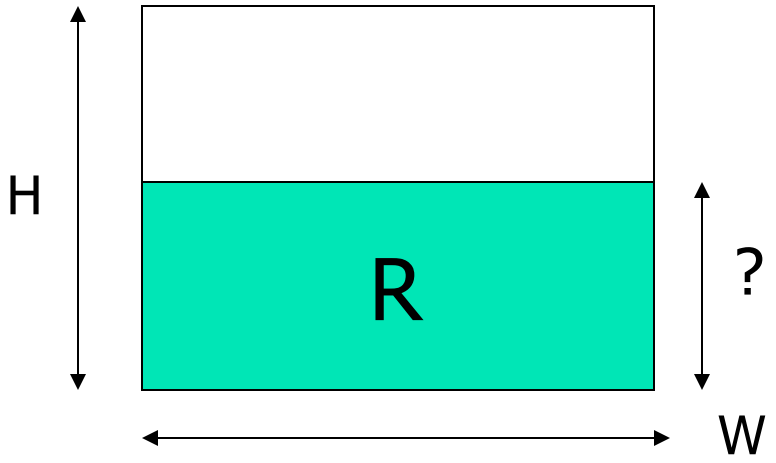
$$R > W / H$$

Match aspect ratios



World window

Aspect Ratio = R

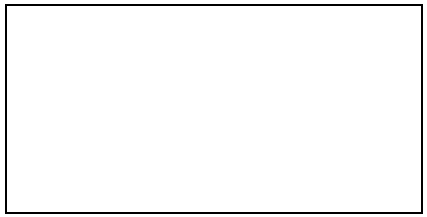


Shrink the viewport height so that

W/new H = R

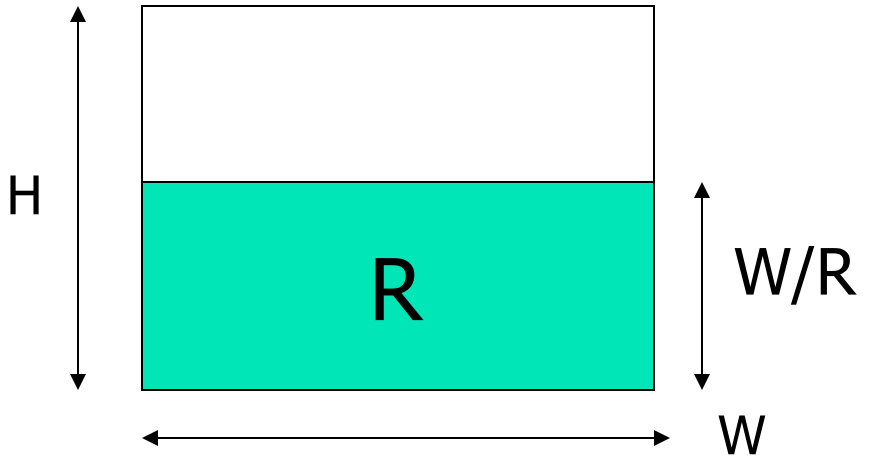
$$R > W / H$$

Match aspect ratios



World window

Aspect Ratio = R



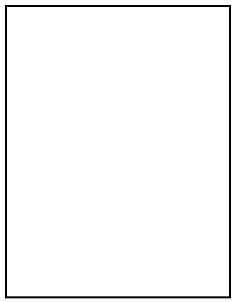
Viewport

Aspect Ratio = W / W/R = R

R > W / H

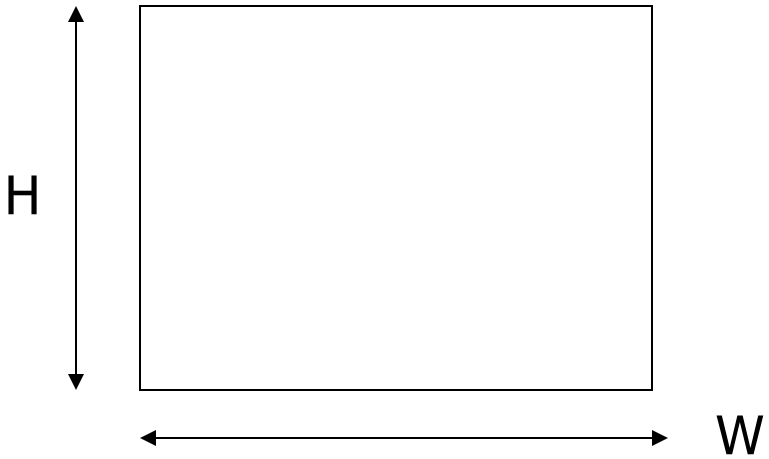
glViewport(0, 0, W, W/R)

Compare aspect ratios



World window

Aspect Ratio = R

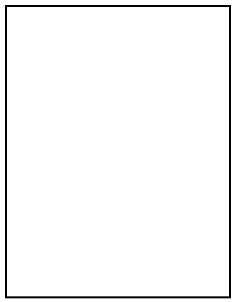


Viewport

Aspect Ratio = W / H

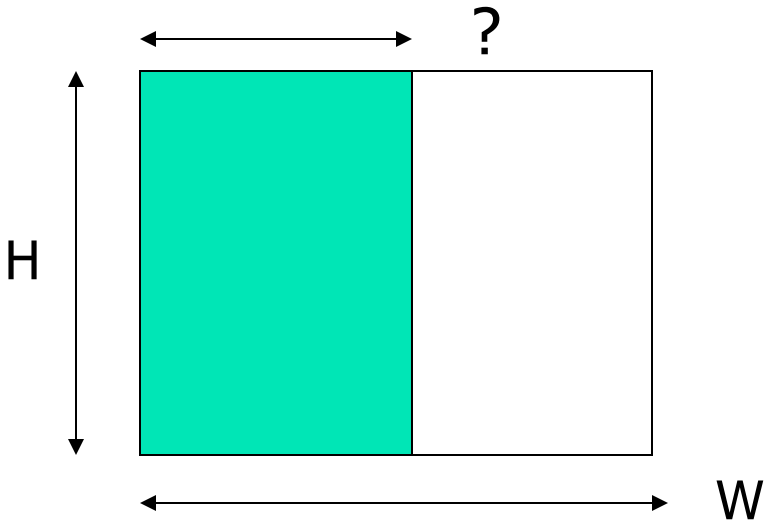
$$R < W / H$$

Match aspect ratios



World window

Aspect Ratio = R

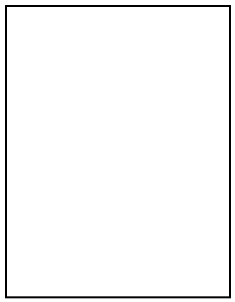


Shrink the viewport width so that

Aspect Ratio = new W / H

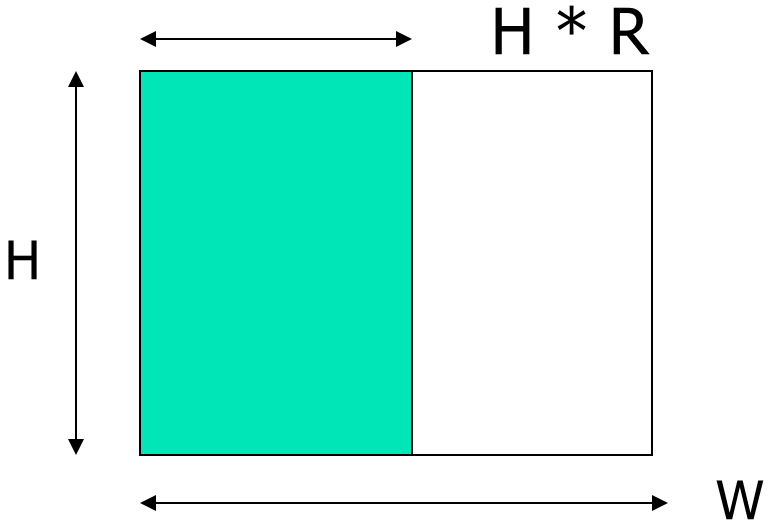
$$R < W / H$$

Match aspect ratios



World window

Aspect Ratio = R

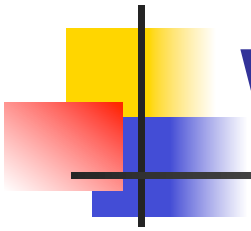


Viewport

Aspect Ratio = H*R / H = R

R < W / H

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



When to call glViewport() ?

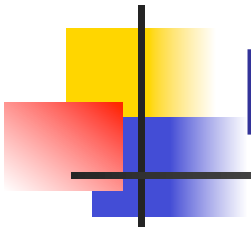
Two places:

- Initialization
  - Default size: same as the window size
- Every time when the user resizes the display window (but GLUT will do that for you)

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

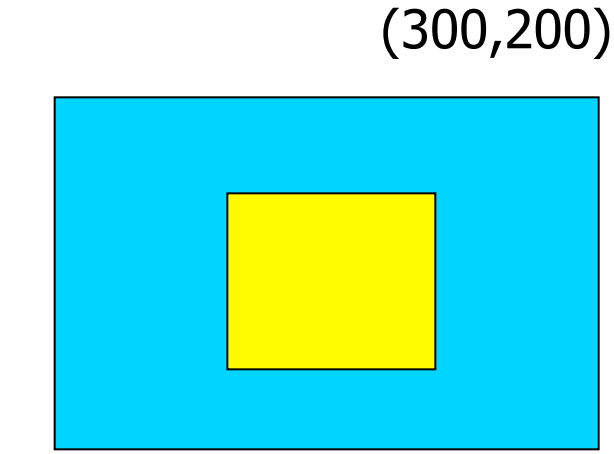
```
Void resize(int W, int H)
{
        glViewport(0,0,W, H);
}
```

This is done by default in GLUT

You can use the call to make
sure the aspect ratio is adjusted as we just discussed.

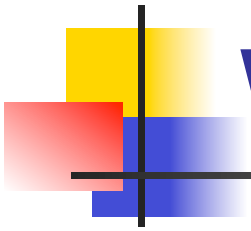
Put it all together

```
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();
}
```



viewport

How big is the quad?



Well, this works too …

```
DrawQuad()
{
    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
    your display window

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