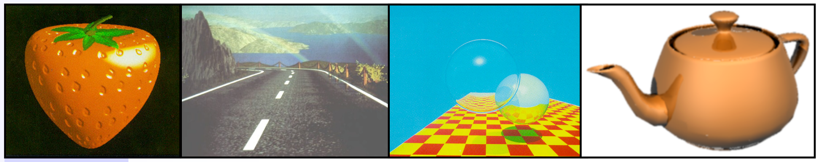
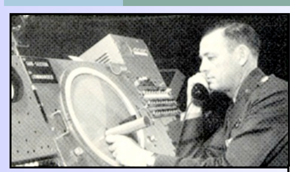


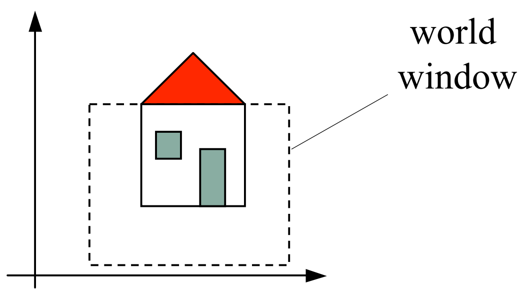
Chap. 4: World Windows, Viewports & Clipping

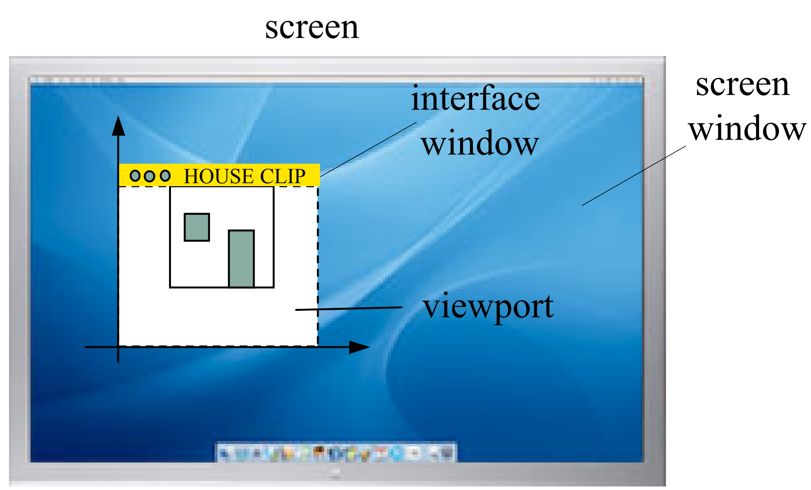


Summary

- Basic definitions: world coordinate systemand screen coordinate system; world window, interface window, and viewport
- Window-to-viewport mapping
- Window-to-viewport transformation in OpenGL
- Rendering pipeline
- 2D Rendering pipeline in OpenGL
- Line clipping: Cohen-Sutherland algorithm
- Polygon clipping
- Clipping in the OpenGL pipeline

Definitions





World Coordinate System (Object Space)

–Space in which the application model is defined; por exemplo **R**^{2}.

–The representation of an object is measured in some physical or abstract units.

–Space in which the object **geometry** is defined.

World Window (Object Subspace)

–Rectangle defining the part of the world we wish to display.

Screen Coordinate System (Image Space)

–Space in which the image is displayed; por exemplo **800x600** pixels.

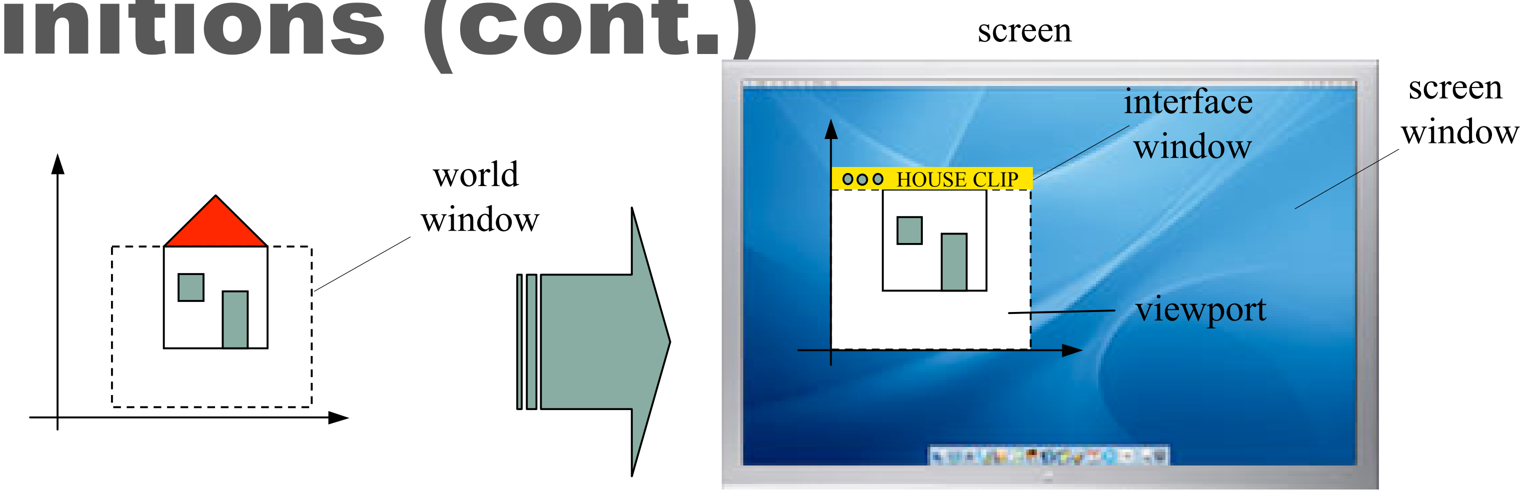
–Usually measured in pixels but could use any units.

–Space in which the object's **raster image** is defined.

Interface Window (Image Subspace)

–Visual representation of the screen coordinate system for windowed displays (coordinate system moves with the interface window)

Definitions (cont.)



Viewing Transformations

–The process of mapping from a world window (world coordinates) to a viewport (screen coordinates) .

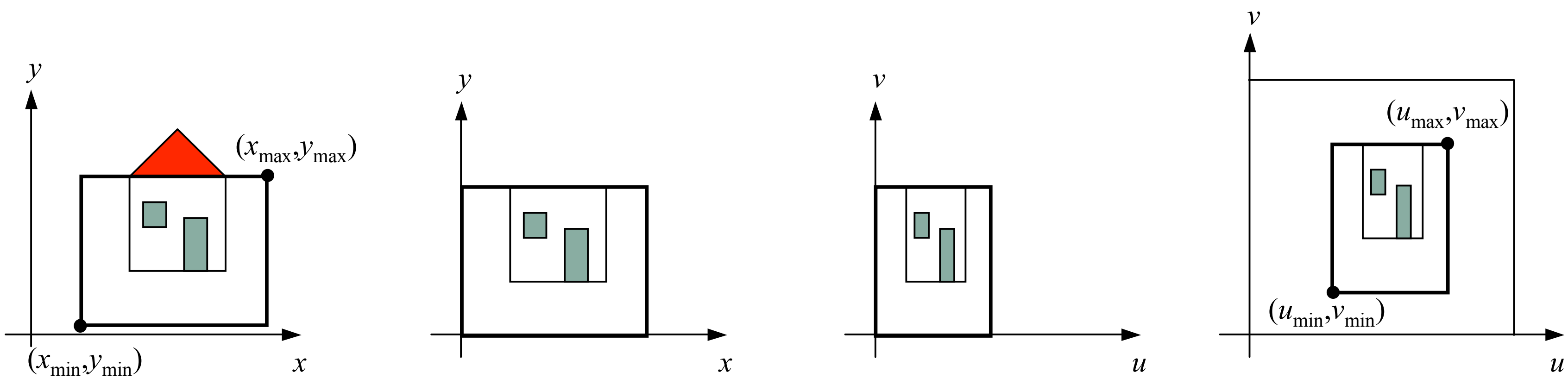
Viewport (Image Subspace)

–A rectangle on the raster graphics screen (or interface window) defining where the image will appear, usually the entire screen or interface window.

–Thus, in principle, the same image can be replicated on different viewports inside the screen or interface window.

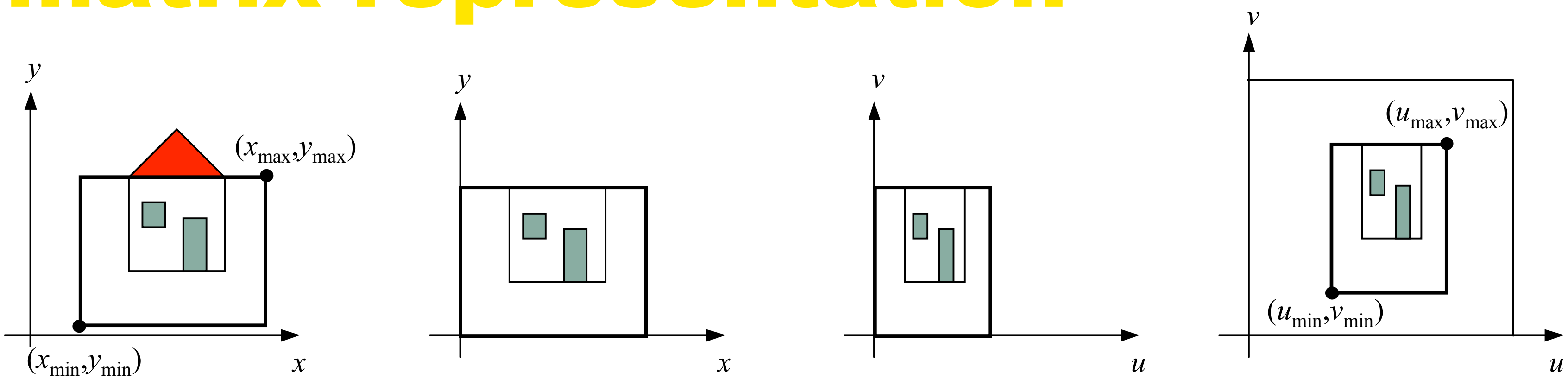
Window-Viewport Mapping

Given a window and viewport, what is the transformation matrix that maps the window from world coordinates into the viewport in screen coordinates? This matrix can be given as a three-step transformation composition as suggested by the following sequence of pictures:



window in world coordinates	window translated to origin	window scaled to size of viewport	translated by (u_{min}, v_{min}) to final position
	$T(-x_{min}, -y_{min})$	$S\left(\dfrac{u_{max}-u_{min}}{x_{max}-x_{min}}, \dfrac{v_{max}-v_{min}}{y_{max}-y_{min}}\right)$	$T(u_{min}, v_{min})$

Window-Viewport Mapping: matrix representation



$T(-x_{min}, -y_{min})$

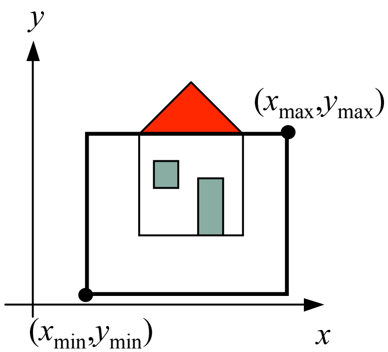
$S(\frac{u_{max}-u_{min}}{x_{max}-x_{min}}, \frac{v_{max}-v_{min}}{y_{max}-y_{min}})$

$T(u_{min}, v_{min})$

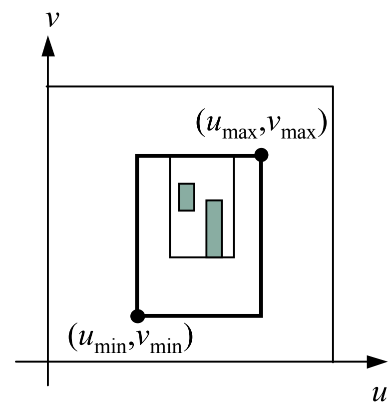
$$M_{wv} = T(u_{min}, v_{min}).S(\frac{u_{max}-u_{min}}{x_{max}-x_{min}}, \frac{v_{max}-v_{min}}{y_{max}-y_{min}}).T(-x_{min}, -y_{min})$$

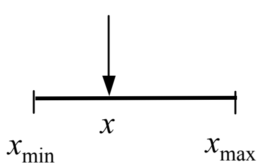
$$= \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & 0 \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

Window-Viewport Mapping:
how is it done?

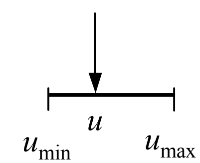


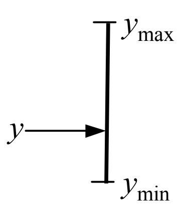
Keeping <u>proportionality</u> in mapping (x,y) to (u,v)



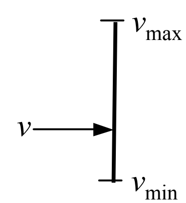


$$\frac{x-x_{\min}}{x_{\max}-x_{\min}}=\frac{u-u_{\min}}{u_{\max}-u_{\min}}\Leftrightarrow u=\overbrace{(x-x_{\min})}^{translation}.\overbrace{\frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}}}^{scaling}\overbrace{+u_{\min}}^{translation}$$

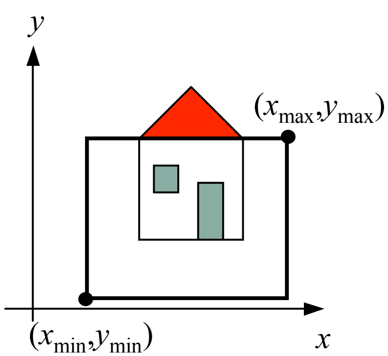


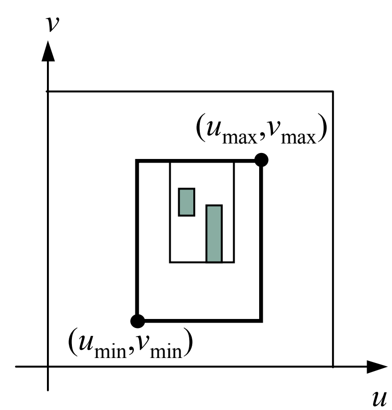


$$\frac{y-y_{\min}}{y_{\max}-y_{\min}}=\frac{v-v_{\min}}{v_{\max}-v_{\min}}\Leftrightarrow v=(y-y_{\min}).\frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}}+v_{\min}$$



Window-Viewport Mapping: example





window(10.0,2.0,40.0,30.0)

viewport(100,50,250,300)

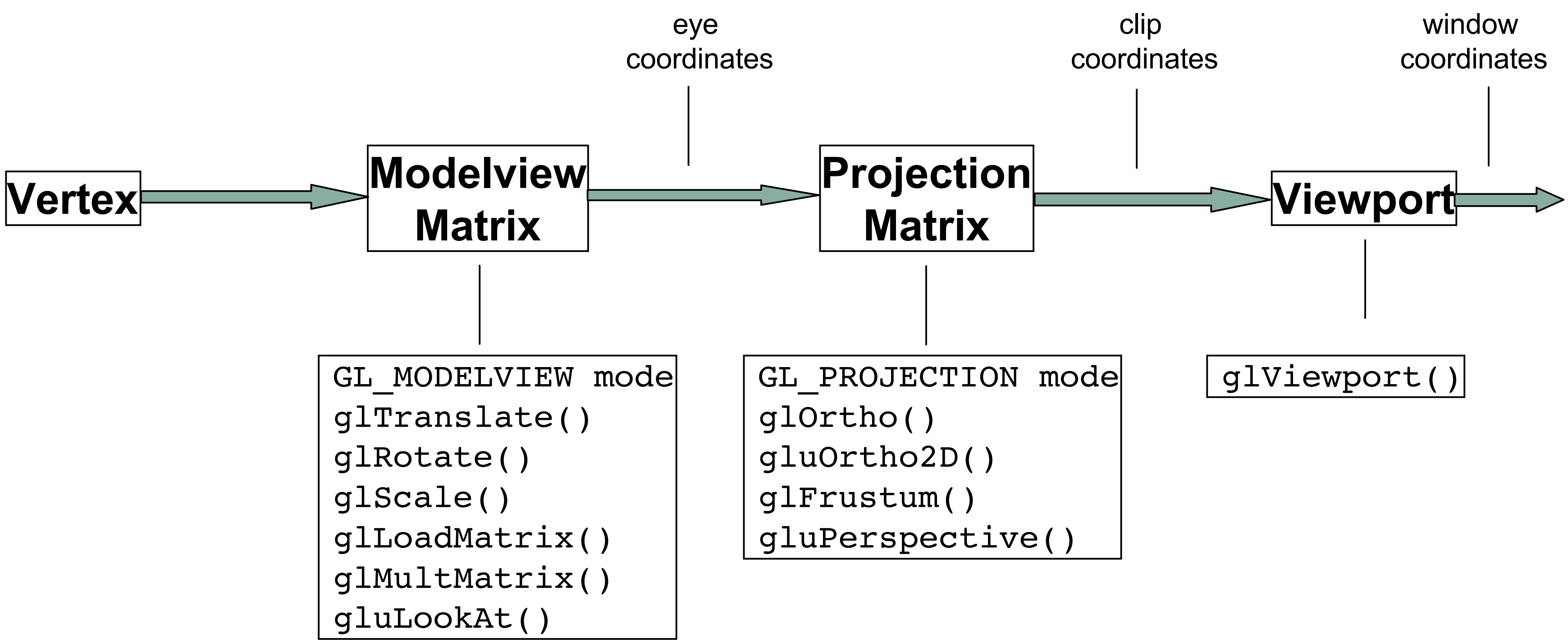
$$u = (x - 10.0).\frac{250-100}{40.0-10.0} + 100 \qquad \lambda_x = \frac{250-100}{40.0-10.0} = 5.0$$

$$v = (y - 5.0).\frac{300-50}{30.0-5.0} + 50 \qquad \lambda_y = \frac{300-50}{30.0-5.0} = 10.0$$

Window-Viewport Mapping: in OpenGL

- `gluOrtho2D(left, right, bottom, top)`
  - Sets up a 2-D orthographic viewing region or *world window*. Defined by two vertical clipping planes `left` and `right` and two horizontal clipping planes `bottom` and `top`.
  - The *world window* by default is (-1,1,-1,1).
  - Defines a 2-D orthographic projection matrix.
  - Sets up the window-viewport mapping, being the viewport defined by the following function:
- `glViewport(x, y, width, height)`
  - Sets up the viewport in the *interface window*, where `x`,`y` specify the lower left corner, and `width`, `height` its dimensions.
  - By default, it uses the whole graphics area of the interface window.
  - There may be various viewports inside the interface window.

Pipeline of OpenGL Transformations



Examples in OpenGL

- A single viewport by default
- A single viewport
- Two viewports

Example 1:
default viewport

```
/* * WV-defaultViewport.cc - Using the default viewport * Abel Gomes   */
#include <OpenGL/gl.h>         // Header File For The OpenGL Library
#include <OpenGL/glu.h>        // Header File For The GLu Library
#include <GLUT/glut.h>         // Header File For The GLut Library
#include <stdlib.h>

void draw(){
          // Make background colour yellow
     glClearColor( 100, 100, 0, 0 );
     glClear ( GL_COLOR_BUFFER_BIT );

          // 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);

          // display rectangles
     glutSwapBuffers();
}                                                 // end of draw()
```

Example 1:
default viewport (cont.)

```
// Keyboard method to allow ESC key to quit
void keyboard(unsigned char key,int x,int y)
{
	if(key==27) exit(0);
}

int main(int argc,  char ** argv)
{
	glutInit(&argc, argv);
		// Double Buffered RGB display
	glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE);
		// Set window size
	glutInitWindowSize( 500,500 );
	glutCreateWindow(”Default viewport spans the whole interface window");
		// Declare the display and keyboard functions
	glutDisplayFunc(draw);
	glutKeyboardFunc(keyboard);
		// Start the Main Loop
	glutMainLoop();
	return 0;
}
```

Example 2:
single viewport

```
/* * WV-singleViewport.cc - Using a single viewport * Abel Gomes  */
#include <OpenGL/gl.h>        // Header File For The OpenGL Library
#include <OpenGL/glu.h>       // Header File For The GLu Library
#include <GLUT/glut.h>        // Header File For The GLut Library
#include <stdlib.h>
void draw(){
          // Make background colour yellow
     glClearColor( 100, 100, 0, 0 );
     glClear ( GL_COLOR_BUFFER_BIT );

          // Sets up 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);

          // display rectangles
     glutSwapBuffers();
}                                                  // end of draw()
```

Example 2:
single viewport (cont.)

```
// Keyboard method to allow ESC key to quit
void keyboard(unsigned char key,int x,int y)
{
     if(key==27) exit(0);
}

int main(int argc,  char ** argv)
{
     glutInit(&argc, argv);
          // Double Buffered RGB display
     glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE);
          // Set window size
     glutInitWindowSize( 500,500 );
     glutCreateWindow(”Single viewport spans the left-bottom interface window quarter");
          // Declare the display and keyboard functions
     glutDisplayFunc(draw);
     glutKeyboardFunc(keyboard);
          // Start the Main Loop
     glutMainLoop();
     return 0;
}
```

Example 3:
two viewports

```
/* * WV-twoViewports.cc - Using two viewports * Abel Gomes  */
#include <OpenGL/gl.h>        // Header File For The OpenGL Library
#include <OpenGL/glu.h>       // Header File For The GLu Library
#include <GLUT/glut.h>        // Header File For The GLut Library
#include <stdlib.h>

void draw(){
          // Make background colour 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 on next page
```

Example 3:
two viewports (cont.)

```
/*  rest of the function draw() */


          // Sets up SECOND viewport spanning the right-top quarter of the interface window
     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()
```

Example 3:

two viewports (cont.)

```
// Keyboard method to allow ESC key to quit
void keyboard(unsigned char key,int x,int y)
{
     if(key==27) exit(0);
}

int main(int argc,  char ** argv)
{
     glutInit(&argc, argv);
          // Double Buffered RGB display
     glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE);
          // Set window size
     glutInitWindowSize( 500,500 );
     glutCreateWindow(”Two viewports spanning the left-bottom and right-top quarters");
          // Declare the display and keyboard functions
     glutDisplayFunc(draw);
     glutKeyboardFunc(keyboard);
          // Start the Main Loop
     glutMainLoop();
     return 0;
}
```

Window-Viewport Mapping: important conclusion

As the world window increases in size

the image in viewport decreases in size

and vice-versa.

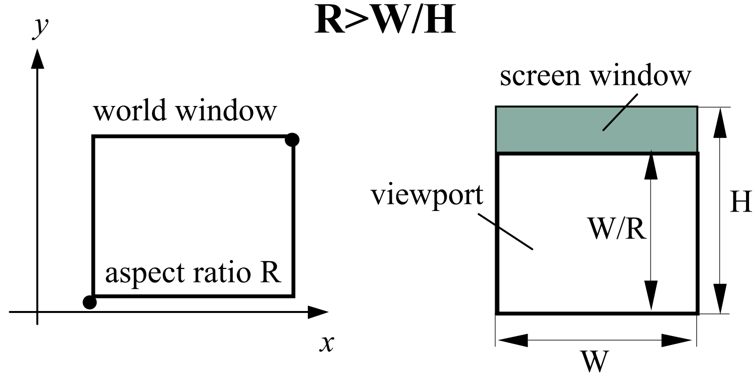
Window-Viewport Mapping: applications

- **Panning**
  - Moving the window around the world
- **Zooming**
  - Reducing/increasing the window size

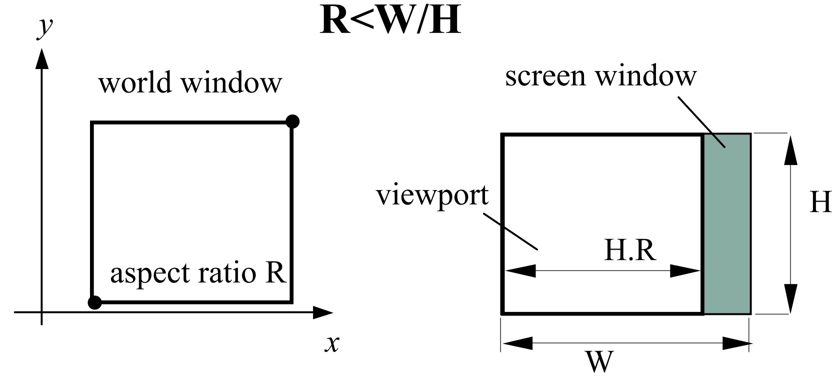
Setting viewport automatically without distortion

- Largest undistorted image that will fit in the screen?
- R = Aspect Ratio of World
- Two situations to be considered:

R>W/H



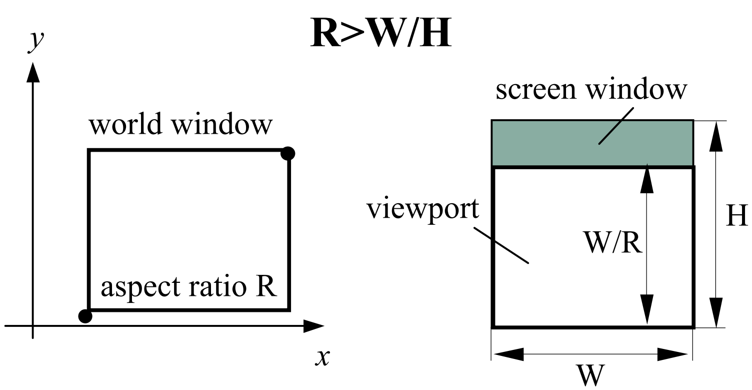
R<W/H

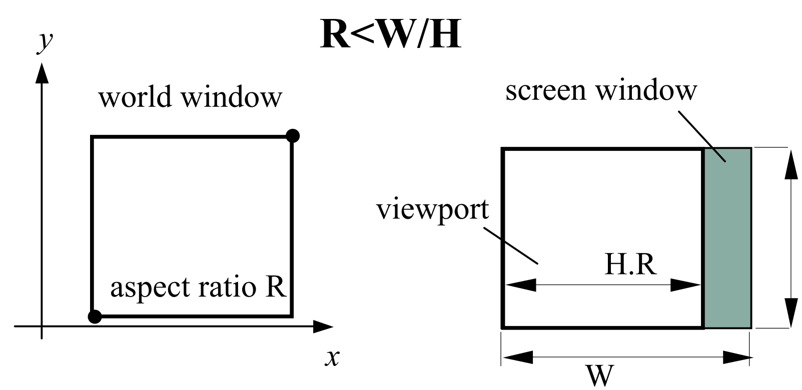


•World window is short and stout compared to screen window.
•Viewport with a matching aspect ration R will extend fully across, but there will be some space unused above/below.
•Therefore, at largest, the viewport will have width W and height W/R.

•World window is tall and narrow compared to screen window.
•Viewport with a matching aspect ration R will extend fully from top tobottom, but there will be some space unused left/right.
•Therefore, at largest, the viewport will have width H.R and height H.

Setting viewport automatically without distortion (cont.)



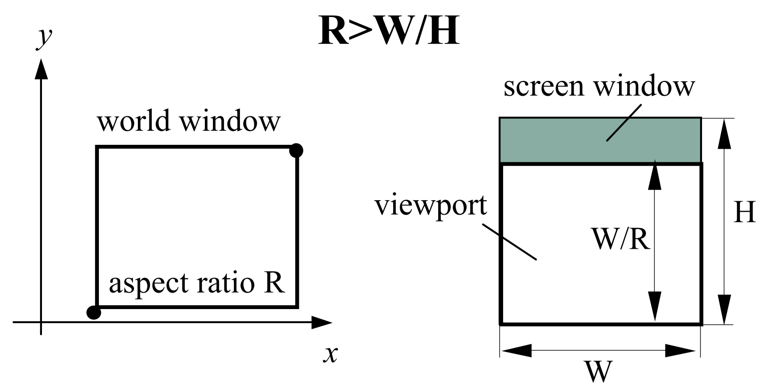



```
glViewport(0,0,W,W/R);
```

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

Example: short window

- If the world window has R=2.0 and the screen has H=200 and W=360, then W/H=1.8.
- Therefore, we fall in first case, and the viewport is set to 180 pixels high and 360 pixels wide.



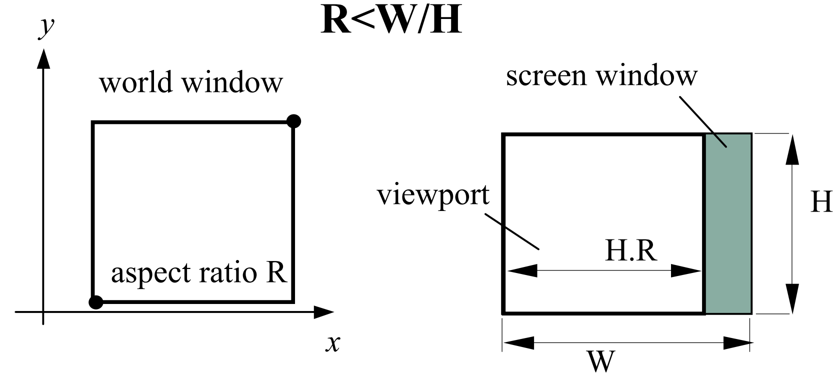

```
glViewport(0,0,W,W/R);

glViewport(0,0,360,360/2);
```

Example: tall window

- If the world window has R=1.6 and the screen has H=200 and W=360, then W/H=1.8.
- Therefore, we fall in second case, and the viewport is set to 200 pixels high and 320 pixels wide.

R<W/H

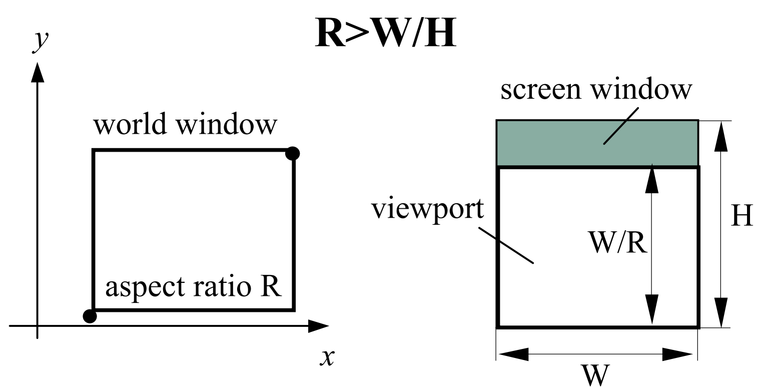


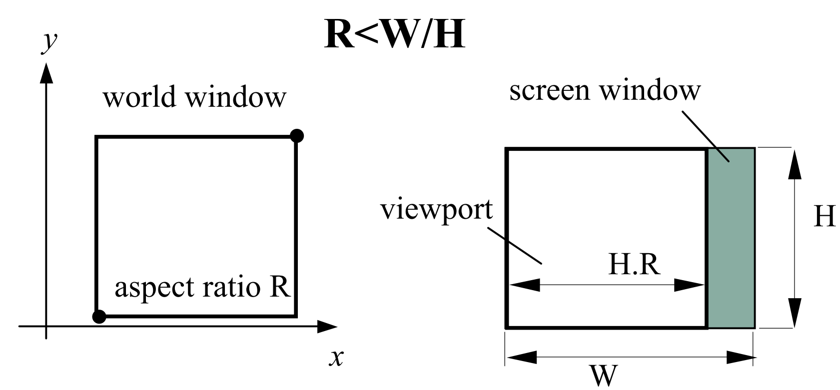

```
glViewport(0,0,H*R,H);

glViewport(0,0,320,200);
```

Example: tall window

- Largest undistorted image that will fit in the screen?
- R = Aspect Ratio of World
- Two situations to be considered:






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

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

Strategy of keeping proportions automatically between window and viewport

- The user may enlarge or reduce the size of a viewport with `w` pixels wide and `h` pixels high by pulling away the right-bottom of its interface window.
- To avoid distortion, we must change the size of the world window accordingly.
- For that, we assume that the initial world window is a square with side length `L`.
- A possible solution is to change the world window whenever the viewport of the interface window were changed. So, the callback `Glvoid reshape(GLsizei w, GLsizei h)` must include the following code :

```
if (w <= h)

    glOrtho2D(-L, L, -L * h/w, L * h/w);

else

    glOrtho2D(-L * w/h, L * w/h, -L, L);
```