Computer Graphics (4731)
Lecture 4: 2D Graphics Systems
(Drawing Polylines, tiling, & Aspect Ratio)

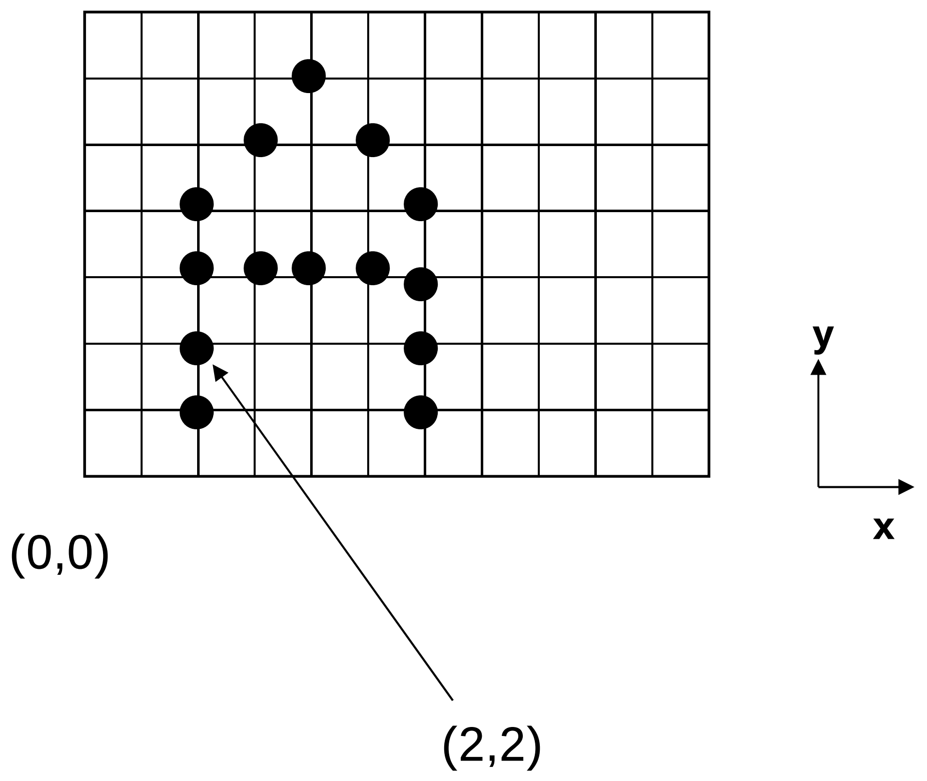
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

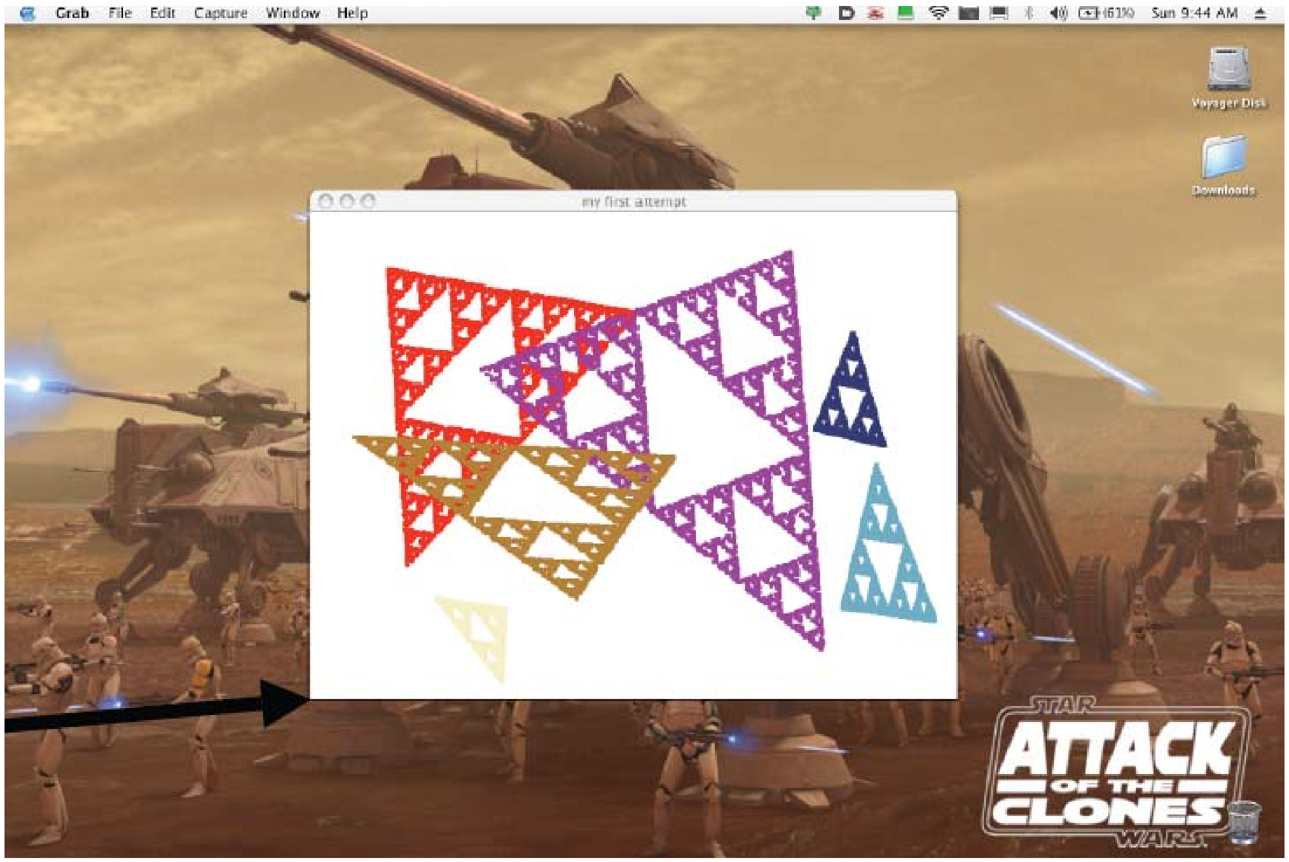
Screen Coordinate System

- Screen: 2D coordinate system (WxH)
- 2D Regular Cartesian Grid
- Origin (0,0): lower left corner (OpenGL convention)
- Horizontal axis – x
- Vertical axis – y
- Pixel positions: grid intersections



Screen Coordinate System

(0,0) is lower left corner of **OpenGL Window.**
NOT lower left corner of entire desktop

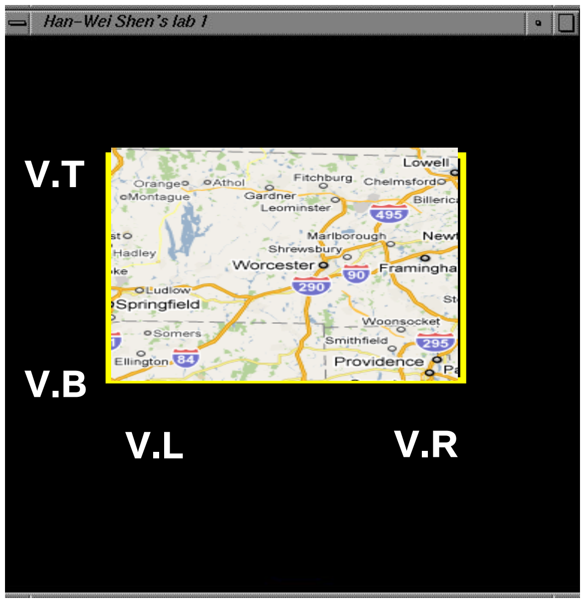

Grab File Edit Capture Window Help
Sun 9:44 AM
Voyager Disk
Downloads
my first attempt
STAR WARS ATTACK OF THE CLONES

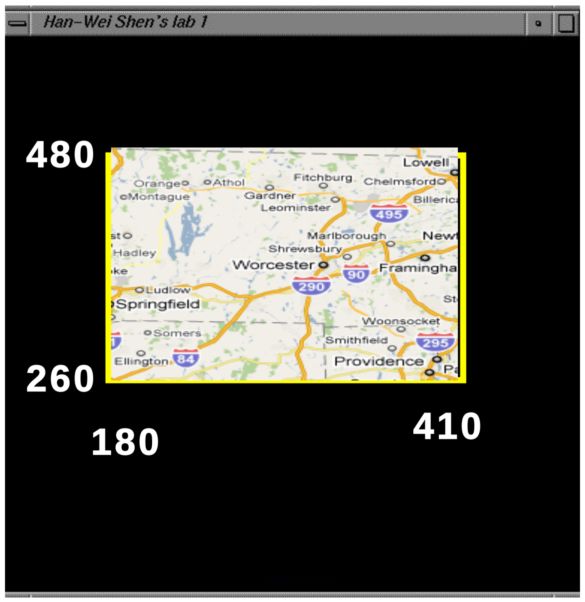
OpenGL's (0,0)

Defining a Viewport

- Can draw to any rectangle (sub-area of screen)
- **Viewport:** Area of screen we want to draw to
- To define viewport

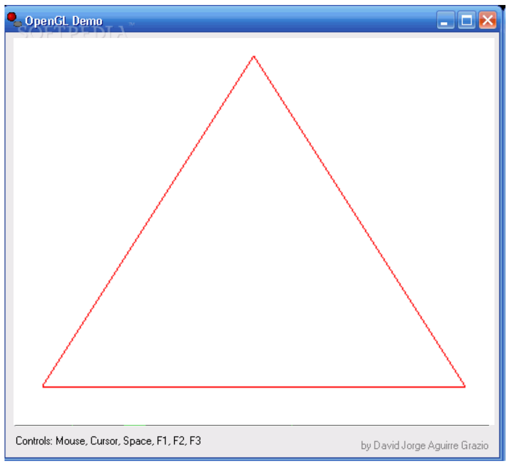
```
glViewport(left, bottom, width, height)
or glViewport(V.L, V.B, V.R – V.L, V.T – V.B)
or glViewport(180, 260, (410 – 180), (480 – 260) )
```





Recall: OpenGL Skeleton

```
void main(int argc, char** argv){
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);    // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );


    // … now register callback functions
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit( );
    glutMainLoop( );
}
```


OpenGL Demo
Controls: Mouse, Cursor, Space, F1, F2, F3
by David Jorge Aguirre Grazio

```
void mydisplay(void){
     glClear(GL_COLOR_BUFFER_BIT);
     glDrawArrays(GL_LINE_LOOP, 0, 3);
     glFlush( );
}
```

Note: default viewport is entire created window

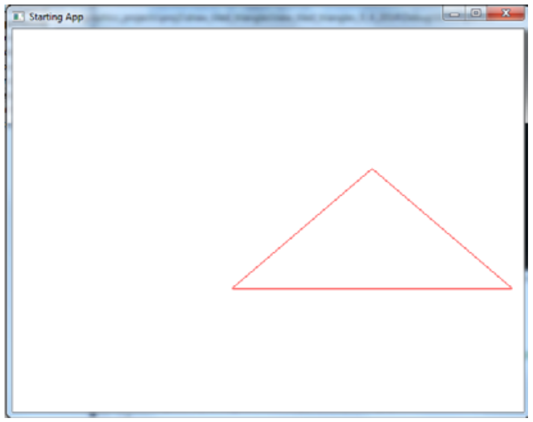
Example: Changing Viewport

How to change viewport to:

Bottom left corner at (100,80)

Width changes to 700, height changes to 300??

```
void main(int argc, char** argv){
   // First initialize toolkit, set display mode and create window
   glutInit(&argc, argv);    // initialize toolkit
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(100, 150);
   glutCreateWindow("my first attempt");
   glewInit( );


   // ... now register callback functions
   glutDisplayFunc(myDisplay);
   glutReshapeFunc(myReshape);
   glutMouseFunc(myMouse);
   glutKeyboardFunc(myKeyboard);

   myInit( );
   glutMainLoop( );
}
```

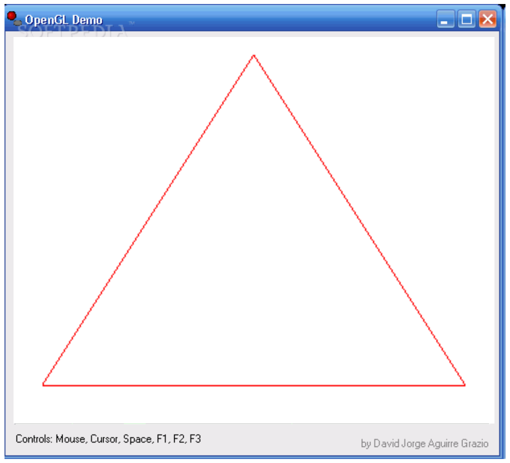


```
void mydisplay(void){
      glClear(GL_COLOR_BUFFER_BIT);
      glViewport(100,80,700,300);
      glDrawArrays(GL_LINE_LOOP, 0, 3);
      glFlush( );
}
```

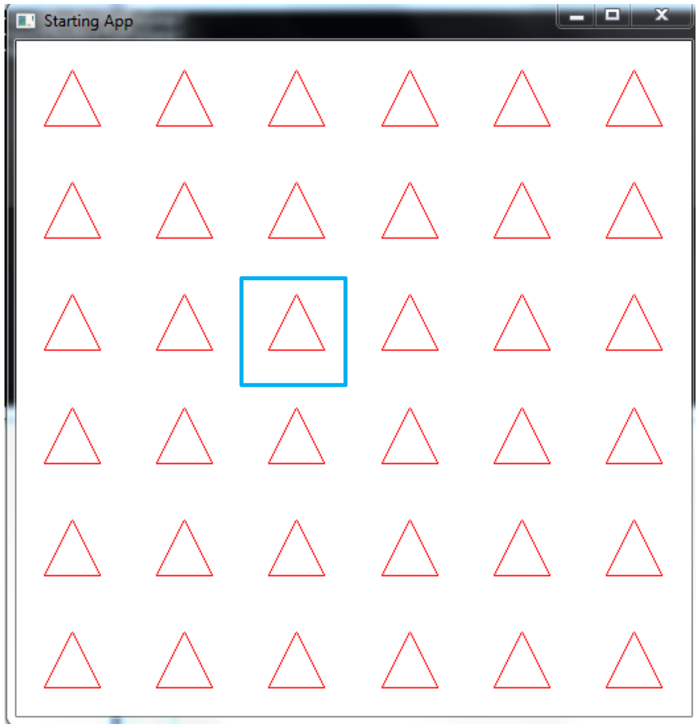
Note: Set desired viewport, then draw

Tiling: Changing Viewport in a Loop

- **Problem:** Want to tile Triangle file on screen
- Solution: change viewport in loop, draw tiles


OpenGL Demo
Controls: Mouse, Cursor, Space, F1, F2, F3
by David Jorge Aguirre Grazio

One world triangle



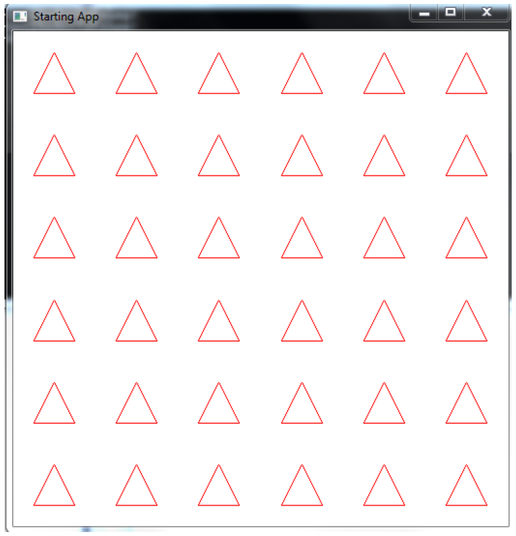
Multiple tiled viewports

Tiling Triangle Code Snippet

- Set viewport, draw into tile in a loop
- Code snippet:

```
float w, h;

w = width / 6;
h = height / 6;

for (int k=0; k<6; k++) {
      for (int m=0; m<6; m++) {
            glViewport(k * w, m * h, w, h);
            glDrawArrays(GL_LINE_LOOP, 0, NumPoints);
      }
}
```

Example: Tiling, Changing Viewport

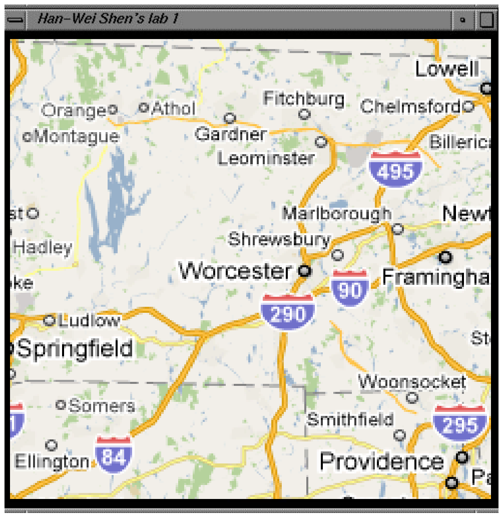


```
void main(int argc, char** argv){
   // First initialize toolkit, set display mode and create window
   glutInit(&argc, argv);   // initialize toolkit
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(100, 150);
   glutCreateWindow("my first attempt");
   glewInit( );

   // … now register callback functions
   glutDisplayFunc(myDisplay);
   glutReshapeFunc(myReshape);
   glutMouseFunc(myMouse);
   glutKeyboardFunc(myKeyboard);

   myInit( );
   glutMainLoop( );
}
```

```
void mydisplay(void){
    glClear(GL_COLOR_BUFFER_BIT);
    float w, h;

    w = width / 6; h = height / 6;

   for (int k=0; k<6; k++) {
       for (int m=0; m<6; m++) {
         glViewport(k * w, m * h, w, h);
         glDrawArrays(GL_LINE_LOOP, 0, NumPoints)
       }
    }
    glFlush( );
}
```

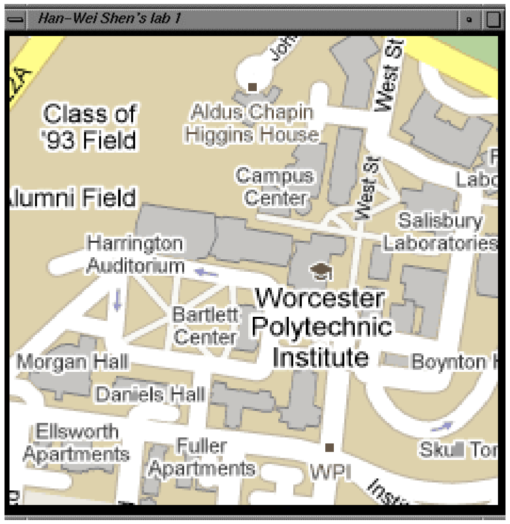
World Coordinate System

- Problems with drawing in screen coordinates:
  - **(x,y) dimensions in pixels:** one mapping, inflexible
  - Not application specific, difficult to use
- **World coordinate:** application-specific
- E.g: Same screen area. Change input drawing (x,y) range



100 pixels = 30 miles

Change World window (mapping)



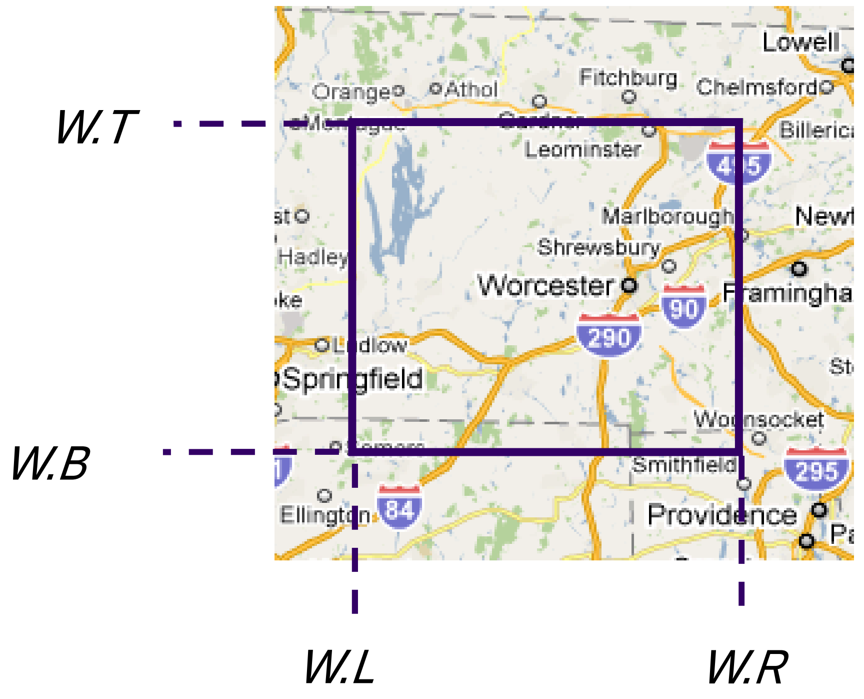
100 pixels = 0.25 miles

Using Window Coordinates

- Would like to:
  - Specify set **source** boundaries (extents) of original drawing in world coordinates (miles, meters, etc)
  - Display **target region** in screen coordinates (pixels)
- Programming steps:
  1. Define world window (original drawing extents)
  2. Define viewport (drawing extents on screen)
  3. Map drawings within window to viewport
- Mapping called ***Window-to-viewport mapping!***

World Coordinate System

- **World Window:** region of **source** drawing to be rendered
- Rectangle specified by world window is drawn to screen
- Defined by (left, right, bottom, top) or (*W.L, W.R, W.B, W.T)*

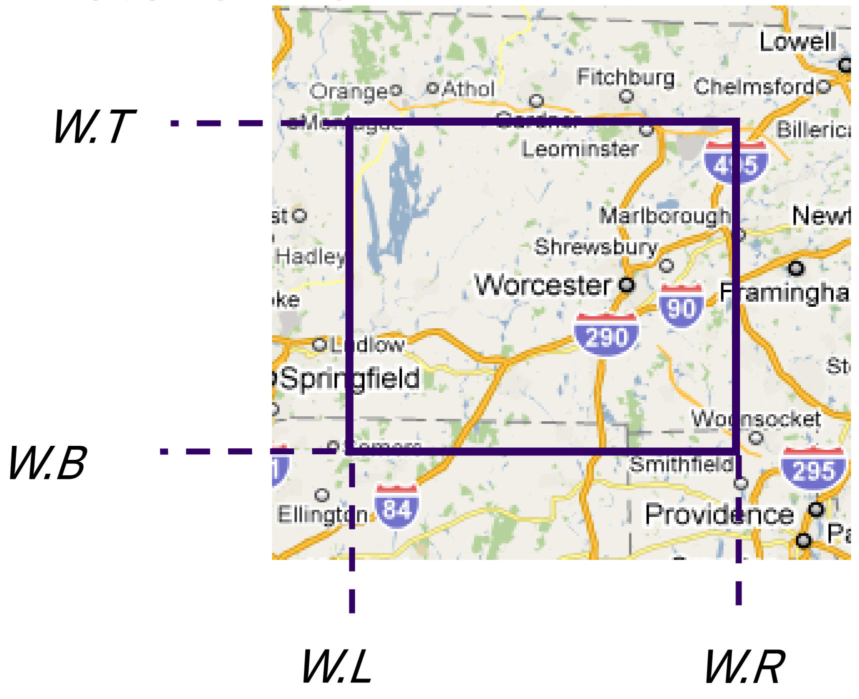


Defining World Window

- `mat4 ortho = Ortho2D(left, right, bottom, top)`

`Or mat4 ortho = Ortho2D(W.L, W.R, W.B, W.T)`

- **Ortho2D** generates 4x4 matrix that scales input drawing
- **Note: Ortho2D** in header file **mat.h**



Drawing

- After setting world window (using ortho2D) and viewport (using glviewport),
  - Draw as usual with **glDrawArrays**

Apply ortho( ) matrix in Vertex Shader

- **One more detail:** Need to pass ortho matrix to shader
- Multiply each vertex by ortho matrix to scale input drawing
- Need to connect **ortho** matrix to **proj** variable in shader

```
mat4 ortho = Ortho2D( W.L, W.R, W.B, W.T );
```

Call Ortho2D in Main .cpp file

```
uniform mat4 Proj;
in vec4 vPosition;

void main( ){
    gl_Position = Proj * vPosition;
}
```

In vertex shader, multiply each vertex with **proj** matrix

Apply ortho( ) matrix in Vertex Shader

1. Include mat.h from book website (ortho2D declared in mat.h)

```
#include "mat.h"
```

2. Connect **ortho** matrix to **proj** variable in shader

```
 mat4 ortho = Ortho2D( W.L, W.R, W.B, W.T );


  ProjLoc = glGetUniformLocation( program, "Proj" );
  glUniformMatrix4fv( ProjLoc, 1, GL_TRUE, ortho );
```

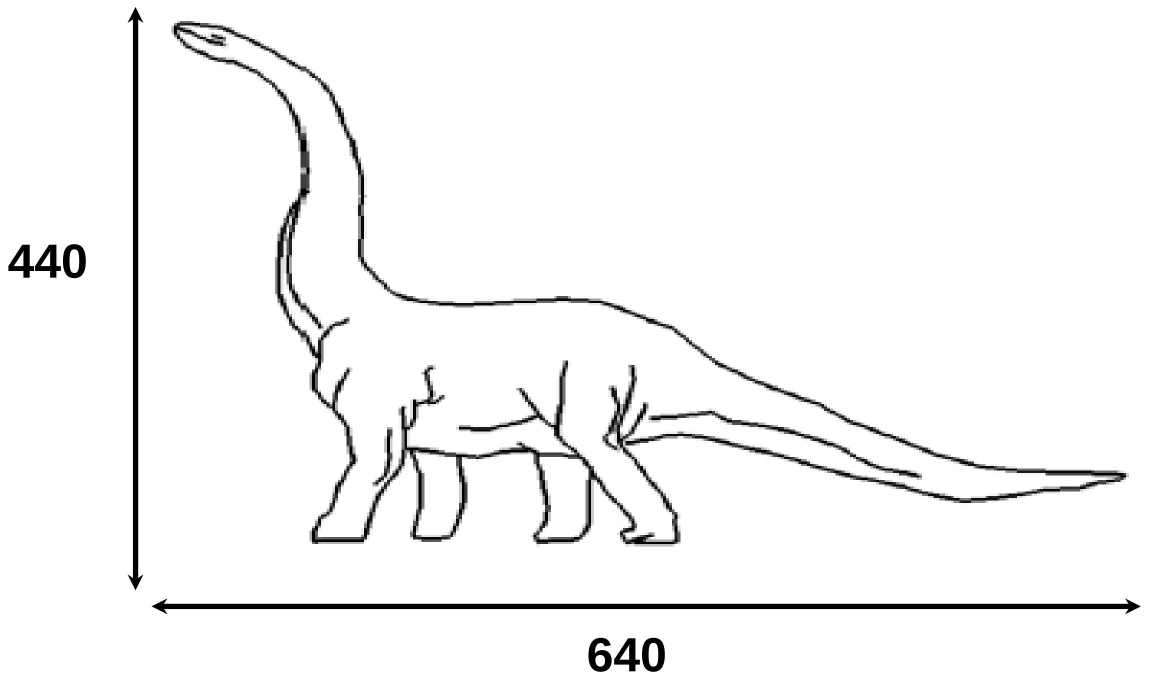
Call Ortho2D in Main .cpp file

```
uniform mat4 Proj;
in vec4 vPosition;

void main( ){
    gl_Position = Proj * vPosition;
}
```

In shader, multiply each vertex with **proj** matrix

Drawing Polyline Files

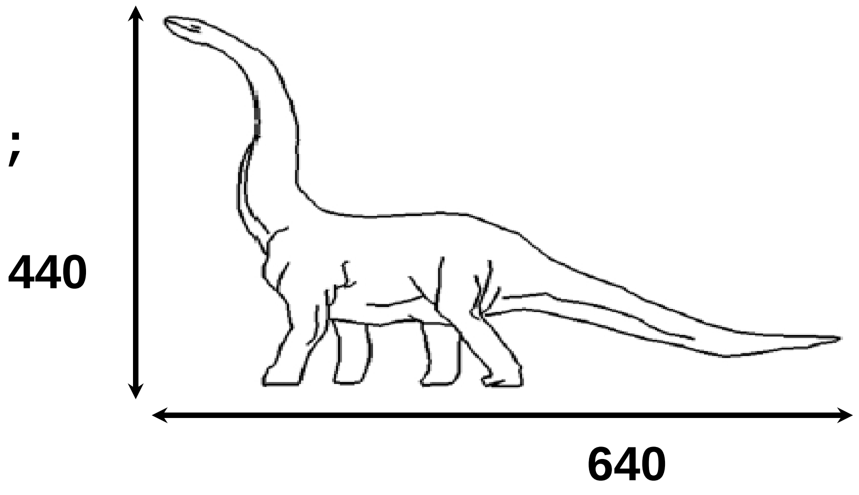
- May read in list of vertices defining a drawing
- **Problem:** want to draw single dino.dat on screen
- **Note:** size of input drawing may vary



Drawing Polyline Files

- **Problem:** want to draw single dino.dat on screen
- Code snippet:

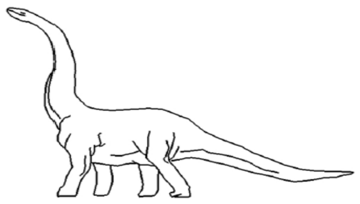
```
// set world window (left, right, bottom, top)
ortho = Ortho2D(0, 640.0, 0, 440.0);

//  now set viewport (left, bottom, width, height)
 glViewport(0, 0, 64, 44);

// Draw polyline fine
  drawPolylineFile(dino.dat);
```



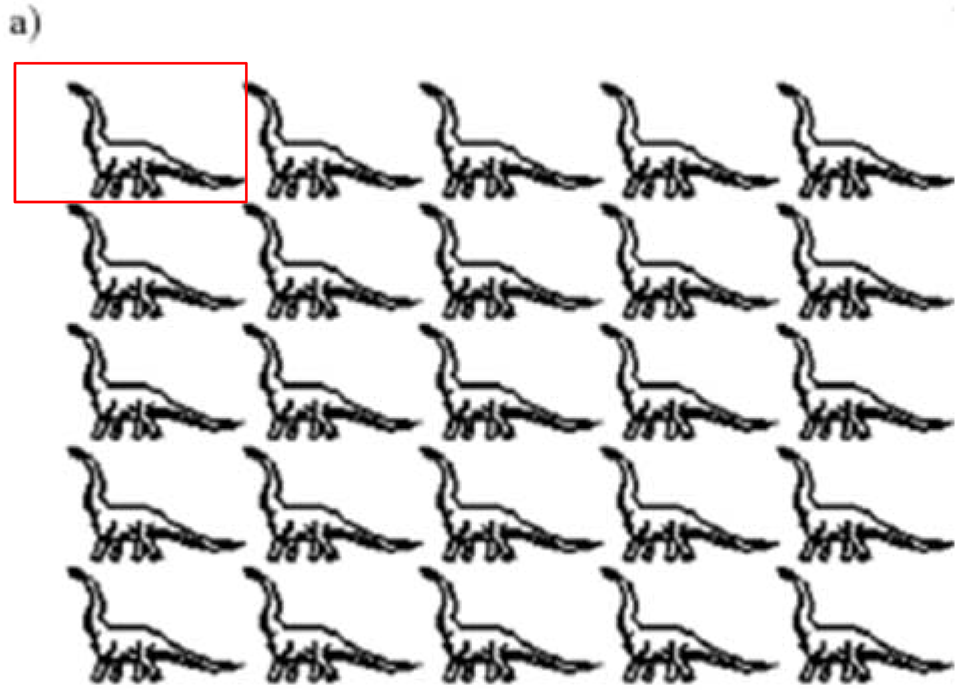
Question: What if I wanted to draw the bottom quadrant of polyline?

Tiling using W-to-V Mapping

- **Problem:** Want to tile polyline file on screen
- Solution: W-to-V in loop, adjacent tiled viewports



One world Window



Multiple tiled viewports

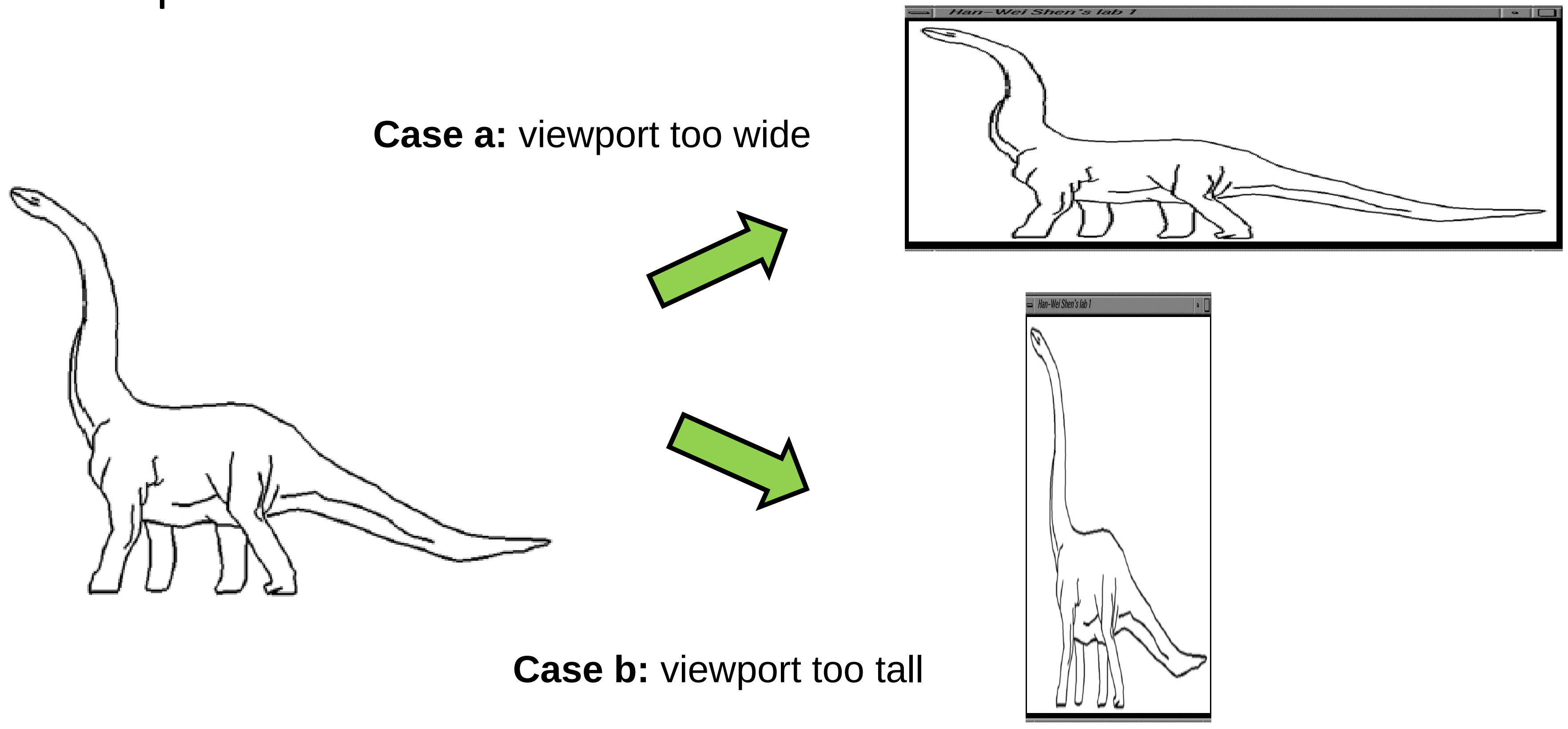
Tiling Polyline Files

- Problem: want to tile dino.dat in 5x5 across screen
- Code snippet:

```
    // set world window
    ortho = Ortho2D(0, 640.0, 0, 440.0);

    for(int i=0;i < 5;i++)
   {
      for(int j = 0;j < 5; j++)
      {   // .. now set viewport in a loop
              glViewport(i * 64, j * 44; 64, 44);
              drawPolylineFile(dino.dat);
     }
   }
```

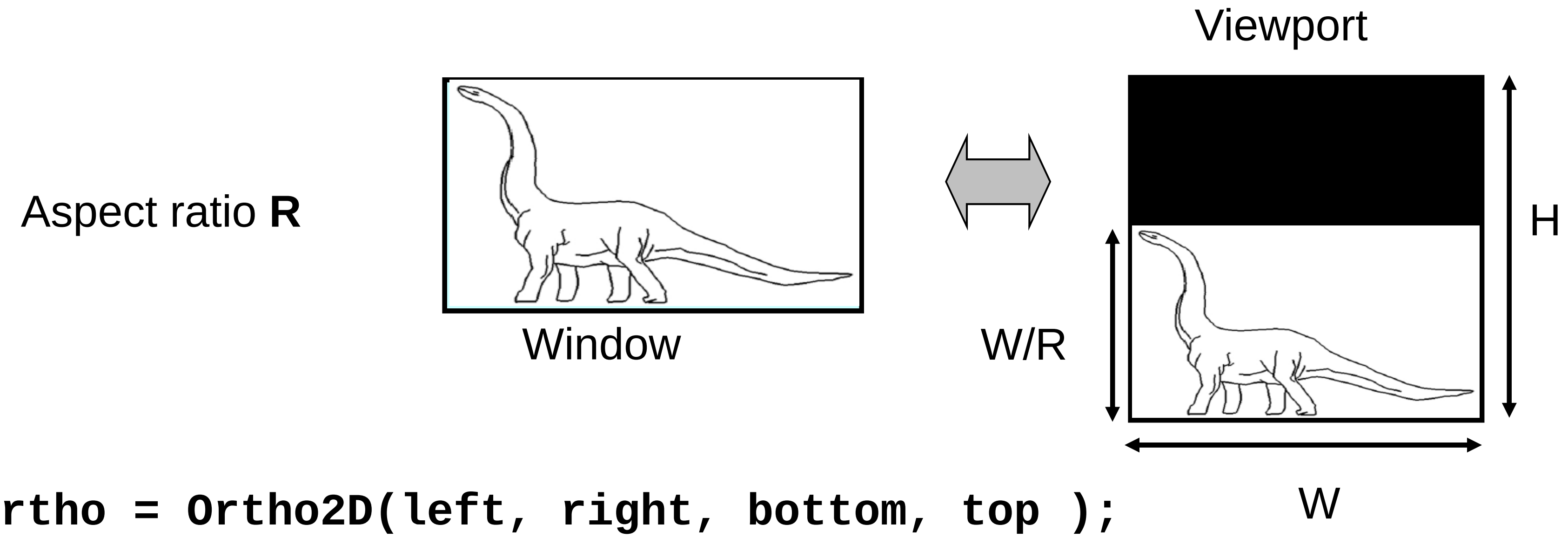
Maintaining Aspect Ratios

- Aspect ratio **R** = Width/Height
- What if window and viewport have different aspect ratios?
- Two possible cases:



What if Window and Viewport have different Aspect Ratios?

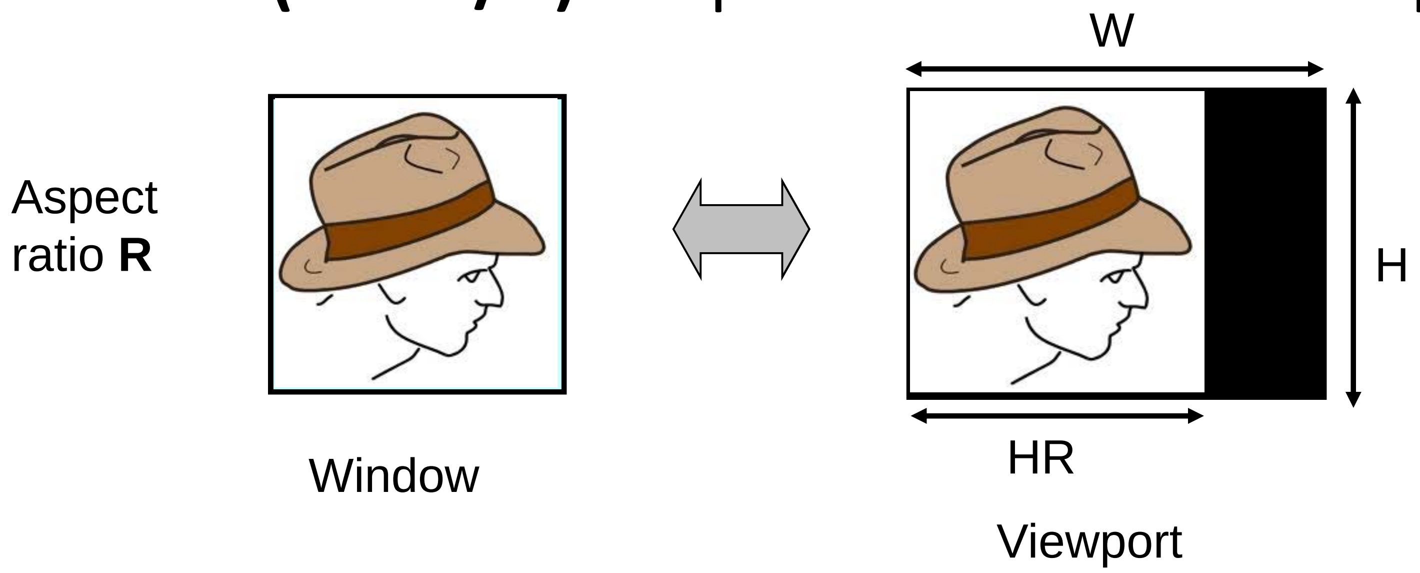
- **R** = window aspect ratio, **W x H** = viewport dimensions
- Two possible cases:
  - **Case A (R > W/H):** map window to tall viewport?



```
ortho = Ortho2D(left, right, bottom, top );
R = (right – left)/(top – bottom);
If(R > W/H)
             glViewport(0, 0, W, W/R);
```

What if Window and Viewport have different Aspect Ratios?

- **Case B (R < W/H):** map window to wide viewport?



```
ortho = Ortho2D(left, right, bottom, top );
R = (right – left)/(top – bottom);
If(R < W/H)
                glViewport(0, 0, H*R, H);
```

reshape( ) function that maintains aspect ratio

```
// Ortho2D(left, right, bottom, top )is done previously,
// probably in your draw function
// function assumes variables left, right, top and bottom
// are declared and updated globally

void myReshape(double W, double H ){
   R = (right – left)/(top – bottom);

   if(R > W/H)
        glViewport(0, 0, W, W/R);
   else if(R < W/H)
        glViewport(0, 0, H*R, H);
   else
        glViewport(0, 0, W, H);  // equal aspect ratios
}
```

References

- Angel and Shreiner, Interactive Computer Graphics, 6th edition, Chapter 9
- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition, Appendix 4