World and Screen Spaces

Lecture 3
CISC 440/640
Spring 2015

COMPUTER GRAPHICS

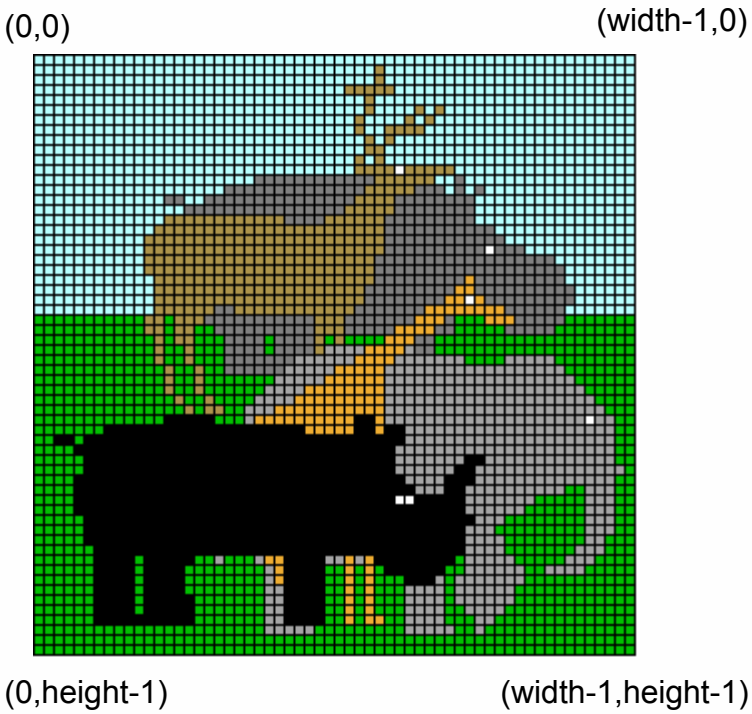
Department of Computer and Information Science

Today's Topic

- How to play with images (load, display, and write)?
- How to map screen to world and world to screen?

Screen Space

- In today's world, graphics are generally presented by establishing colors for a set of discrete samples called "pixels"
- Pixels are displayed on screen in windows
- Logically, pixels are addressed as two-dimensional arrays, whose indices are called "Screen-space" coordinates



OpenGL Tools Available

Typical OpenGL code to establish a window:

```
glutInitWindowSize(400,400);
glutInitWindowPosition(100,100);
```

Code to set up a viewport:

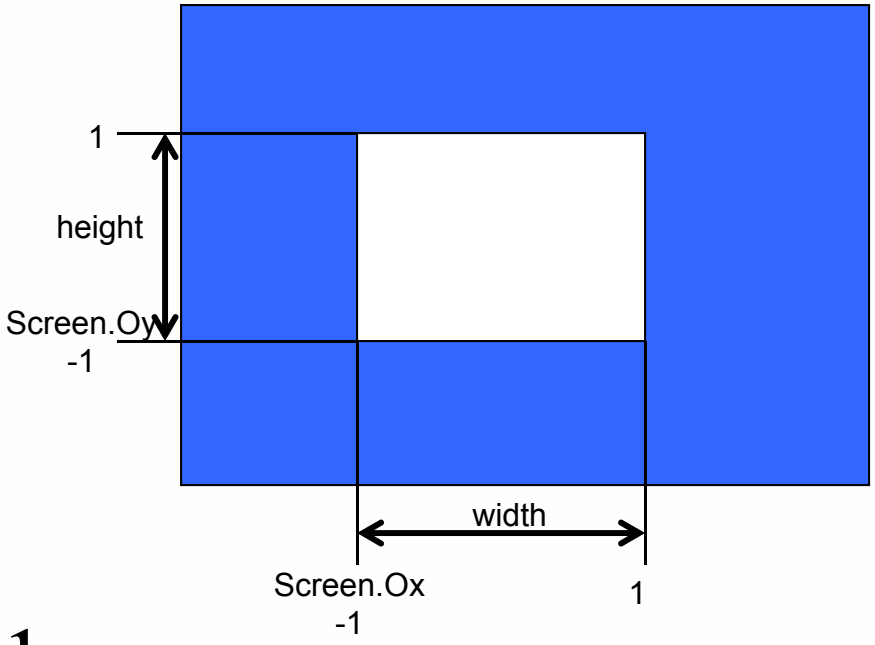
```
glViewport(0, 0, (GLsizei) w, (GLsizei) h);
```

From screen coordinate to world

- Same approach

$$\frac{S.x - S.Ox}{width} = \frac{x - (-1)}{1 - (-1)}$$

- Solve for x



$$x = 2 * \frac{S.x - S.Ox}{width} - 1$$

Beyond [-1, 1]: A Bigger World

To establish a world space coordinate system:

glOrtho2D(world.l, world.r, world.b, world.t);

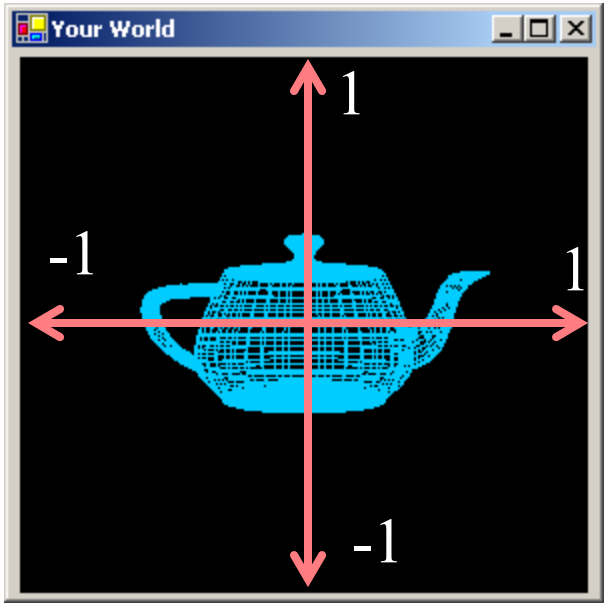
$$\frac{S.x - S.Ox}{width} = \frac{x - W.l}{W.r - W.l}$$

$$x = (W.r - W.l) * \frac{S.x - S.Ox}{width} + W.l$$

- How do we get S.x? How do we get S.Ox? How do we get width? Why do we need it?
- Resizing, multiple viewports, etc. glGetIntegerv(GL_VIEWPORT, viewport);

Normalized Device Coordinates

- Rather than construct a single mapping from world-to-screen, we will instead introduce an intermediate "rendering-space" and compose this two mappings.
- This "rendering-space" is what we've been using a our default to this point.
- This space is called Normalized Device Coordinates (NDC)
- Sometimes called "canonical screen" space



Why introduce NDC?

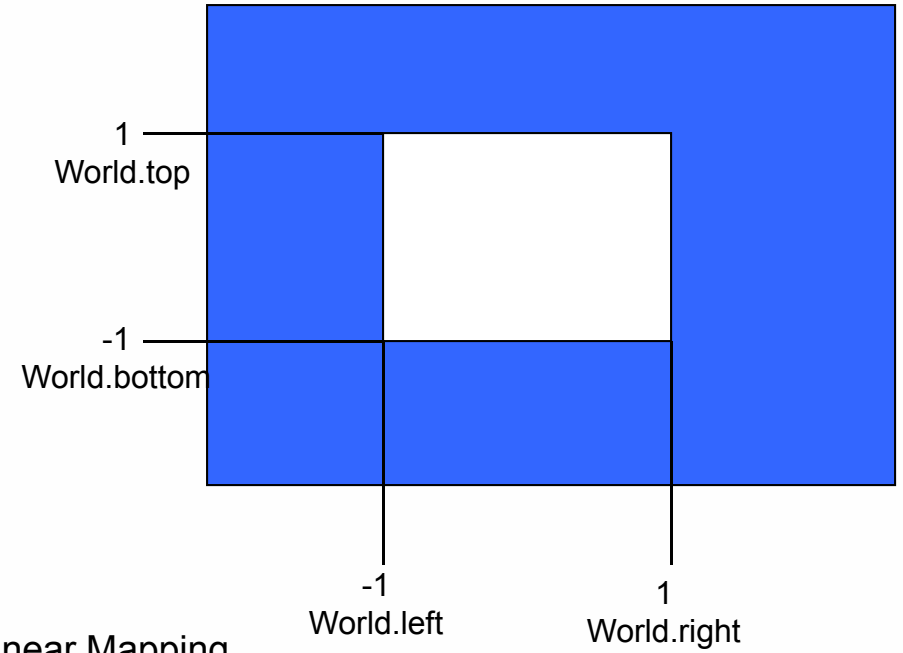
- Easy to convert NDC to a fixed-point representation. Why?
- Simplifies many rendering operations and makes them more amenable to a H/W implementation (Clipping, Computing coefficients used in interpolation)
- Separates the bulk of geometric processing from the specifics of rasterization (sampling)

World to Screen: Rendering

For a World to NDC mapping:

- We know

$$\frac{x-(-1)}{1-(-1)} = \frac{W.x - W.l}{W.r - W.l}$$



$$x = 2\frac{W.x - W.l}{W.r - W.l} - 1$$

$$x = A * W.x + B$$

That's just a Linear Mapping.
We know of a compact notation for such mappings

Where:

$$A = \frac{2}{W.r - W.l} \qquad B = -\frac{W.r + W.l}{W.r - W.l}$$

Mapping as a Matrix Multiplication

- Since our world space to rendering space mappings are linear, they can be accomplished via a matrix multiplication

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{W.r - W.l} & 0 & -\frac{W.r + W.l}{W.r - W.l} \\ 0 & \frac{2}{W.t - W.b} & -\frac{W.t + W.b}{W.t - W.b} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} W.x \\ W.y \\ 1 \end{bmatrix}$$

From NDC to screen coordinates

- Same approach

$$\frac{S.x - S.Ox}{width} = \frac{x - (-1)}{1 - (-1)}$$

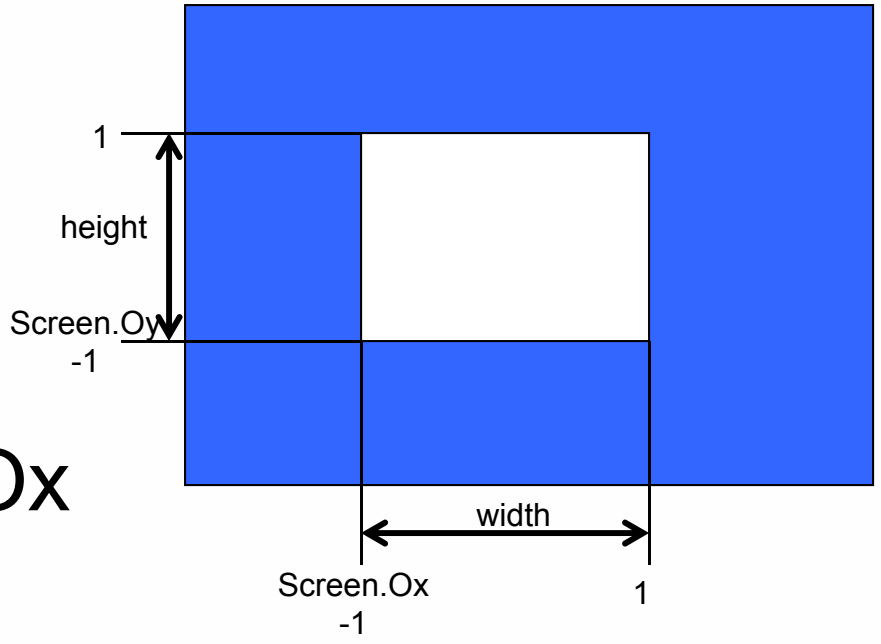
- Solve for S.x

$$S.x = \frac{x+1}{2} width + S.Ox$$

$$S.x = Ax + B$$

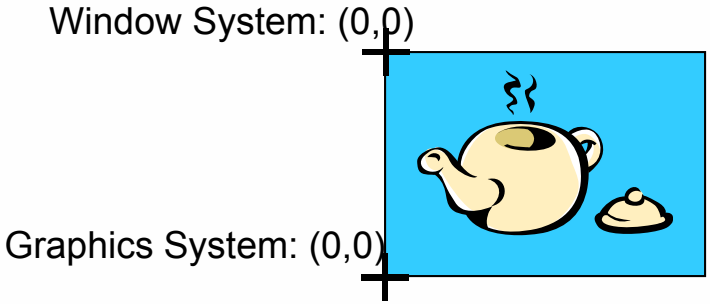
where

$$A = \frac{width}{2} \qquad B = \frac{width}{2} + S.Ox$$



Mapping from world to screen

- The NDC-to-screen mapping "could" be implemented as a matrix transform, but, generally, it is not (dedicated Hardware)
  - World-to-NDC floating point
  - NDC-to-screen fixed-point



- Inserts an opportunity to compensate for differences in coordinate frames
- OpenGL provides a lot of helpful tools, but not enough. We often need to construct a screen-to-world mapping

Screen-to-World in Practice

```
void initialize( )
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h); // NDC-to-screen mapping
    glMatrixMode(GL_PROJECTION); // specify camera model
    glLoadIdentity();
    gluOrtho2D(world.l, world.r, world.b, world.t);  // World-to-NDC
    mapping
    glMatrixMode(GL_MODELVIEW); // specify object transformation
    glLoadIdentity();
    glGetIntegerv(GL_VIEWPORT, viewport);     // squirrel away for later
}
```

The "GLUTs" of the program

```
void main(int argc, char** argv)
{
    int w = 256, h = 256;
    glutInit(&argc, argv);                          // gives GLUT a shot at the  command-line arguments
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(w,h);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Image Viewer");       // Set's the text on the Title Bar
    glutDisplayFunc(display);                                   // routine used to draw the screen
    glutMouseFunc(onMouseButton);                               // routine used to process mouse
    glutKeyboardFunc(onKeyPress);                               // routine used to process keyboard presses
    glutReshapeFunc(resize);                                    // routine that resets viewport on a resize event
    initialize();                                               // user routine to initialize state
    glutMainLoop();
}
```

Lots of global variables and classes

```
class OpenGLImage;
class ImageWindow;
typedef struct {
        double l, r, b, t;
} Extent;

Extent world = {-1, 1, -1, 1};
GLint viewport[4];

void display() {
    img_win->Draw();
    glutPostRedisplay();
}
```

(and an old Friend)

```
void ImageWindow::Init( int w, int h, Extent world)
{
    width_ = w;
    height_ = h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(world.l, world.r, world.b, world.t);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glGetIntegerv(GL_VIEWPORT, viewport);
    if (firstTime) {
        glEnable(GL_TEXTURE_2D);
        glGenTextures(MAX_IMAGES,
imageTextures);
        firstTime = false;
    }
}
```

Initialize

- Executed before any other OpenGL calls, called when major changes occur:

```
void initialize()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(world.l, world.r, world.b, world.t);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glGetIntegerv(GL_VIEWPORT, viewport);
}
```

- Note: by default GLUT set's up a viewport, we only need to mess with it if the window size changes.

An Updated Old Friend

```
void ImageWindow::Draw() {
    glClearColor(0, 0, 0, 1);
    glClear( GL_COLOR_BUFFER_BIT );
    for ( int i = 0; i < nextTexID; i++ )  {
        imageList[i]->draw( );
    }
   glutSwapBuffers();
}
```

Texture Map the Quad

```
void OpenGLImage::draw() {
    // Clear the transformation matrix and load the new one:
    glLoadIdentity();
    updateXform();
    glLoadMatrixd(glMatrix_);
    // Bind the texture associated with this QUAD
    glBindTexture(GL_TEXTURE_2D, textureID_);

    // Draw the QUAD: specify texture coordinates for each vertex
    glBegin(GL_QUADS); {
        glTexCoord2d(0, 0);
        glVertex2d(-1,-1);
        glTexCoord2d(((double)width_-1)/texWidth_, 0);
        glVertex2d(1, -1);
        glTexCoord2d(((double)width_-1)/texWidth_, ((double)height_-1)/texHeight_);
        glVertex2d(1, 1);
        glTexCoord2d(0, ((double)height_-1)/texHeight_);
        glVertex2d(-1, 1);
    }
    glEnd();
}
```

Recap all the GLUT stuff

```
void main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(400,400);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Julia Set (a.k.a. Whoville)");
    glutDisplayFunc(display);
    glutMouseFunc(onMouseButton);
    glutKeyboardFunc(onKeyPress);
    glutReshapeFunc(resize);

    initialize();
    c.re = 0.109;          c.im = 0.603;
    glutMainLoop();
}
```

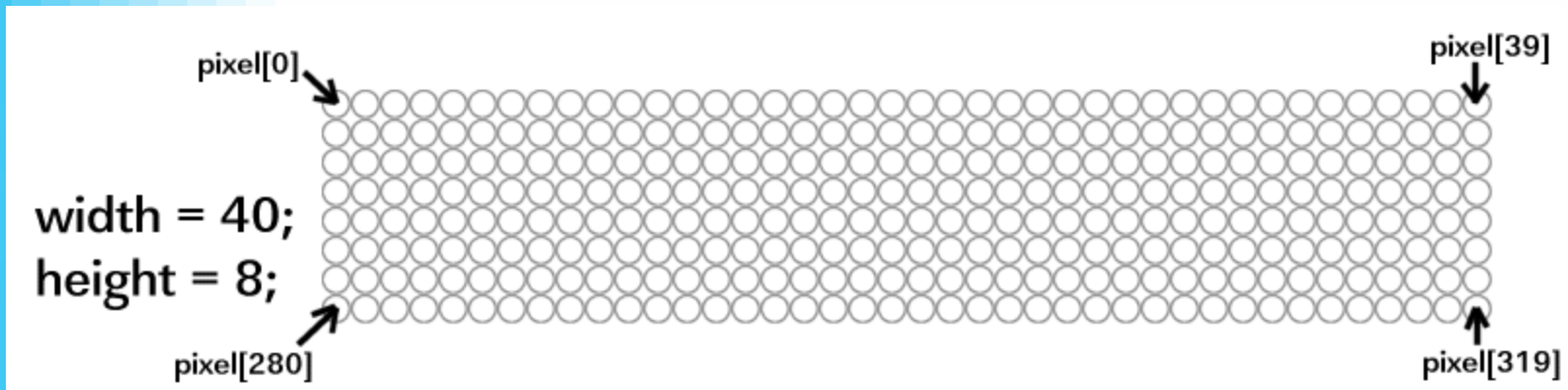
Step1: Loading Image Using CxImage

- An API for image loading/writing/processing
- You will need that for your 1st assignment
- Store an image as an array of pixels (what are pixels?)

```
int readFileCxImage(const char *filename1, int &width, int &height)
{
    string ext = FindExtension(string(filename1));

    if (ext == "") return FALSE;
    int type = FindType(ext);

    CxImage * image = new CxImage();

    image->Load(filename1, type);
      //Initialize the image info structure and read an image.

      // Set the data size to accomodate this new image.
    width = image->GetWidth();
    height = image->GetHeight();


      // This is the method for reading pixels that compiles and works,
      // as opposed to GetImagePixels or GetOnePixel, which wouldn't compile.

    if ( file_data )
        delete [] file_data;
    file_data = new GLubyte[width*height*3];

    //flip rgb and topb
    for (unsigned int j=0; j<height; j++) {
       for (unsigned int i=0; i < width; i++) {
           //  data ranges 0 to 256
           // Swap data vertically, to match NM convention.
           file_data[j*width*3+i*3+0]= image->GetPixelColor(i, height - j).rgbRed;
           file_data[j*width*3+i*3+1]= image->GetPixelColor(i, height - j).rgbGreen;
           file_data[j*width*3+i*3+2]= image->GetPixelColor(i, height - j).rgbBlue;
        }
    }
    return 0;
}
```

How to Display an Image?

- Load the image
  - Use C++ packages like CxImage
- What is an Image?
  - An image is an array of pixels
  - Each pixel has

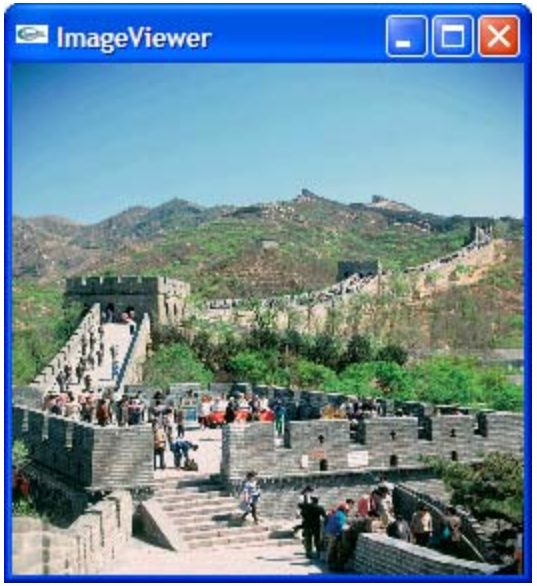


Step 2: Displaying
Method 1: OpenGL Imaging

- glDrawPixels() – Writes an array of pixels to the framebuffer
- glReadPixels() – Reads an region of the framebuffer into an array of pixels in main memory
- glCopyPixels() – Copies a region from one part of the framebuffer to another
- glRasterPos*() – Sets the current drawing position for glDrawPixels() and destination position of glCopyPixels()

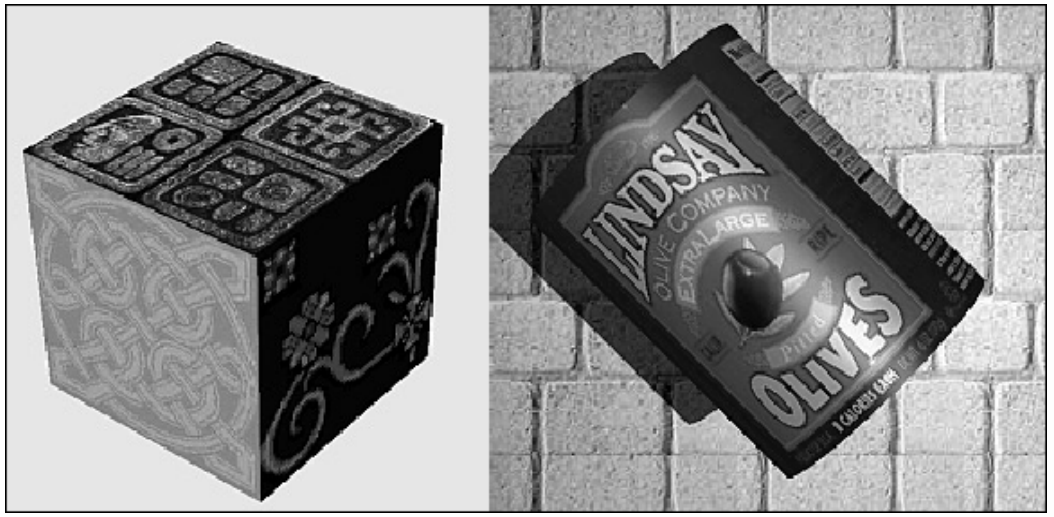
Method 2: Texture Mapping

- First, draw a quad
- Second, texture map the quad with an image
- How?
  - Generate a texture
  - Enable GL_TEXTURE
  - Specify texture/vertex correspondences
  - Call glVertex()



What is Texture Mapping?

- Spatially-varying modification of surface appearance at the pixel level
- Characteristics
  - Color
  - Shininess
  - Transparency
  - ***Bumpiness***
  - Etc.



- **Sprite** when on image-aligned polygon

from Hill

Texture mapping applications: Billboards

from Akenine-Moller & Haines

OpenGL: Texture coordinates at vertices

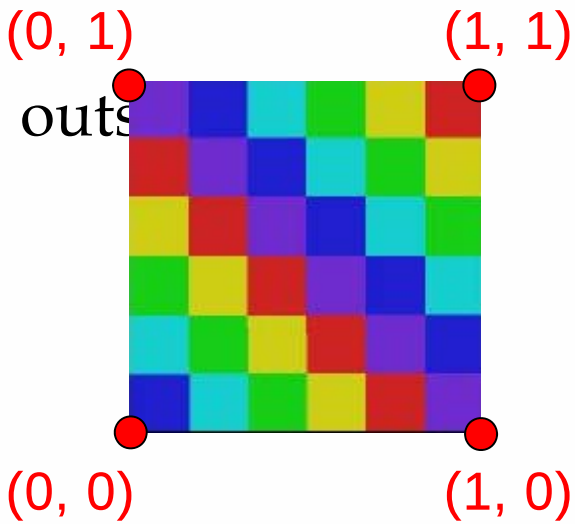
- To draw textured shape, texturing must first be enabled: **`glEnable(GL_TEXTURE_2D)`**
- Designate texture image with **`glTexImage2D()`**
  - Width, height must be powers of 2 (plus 2 if border is used)
  - Only one texture current; faster to change textures by preloading all and switching with **`glBindTexture()`** rather than reloading each time
- Assign texture coordinates `(s, t)` to vertices with **`glTexCoord()`**
  - Similar to **`glColor()`** command—sets a property for subsequent vertices that holds until it is changed
- To be able to use glColor(), texturing must be disabled: **`glDisable(GL_TEXTURE_2D)`**

OpenGL: Corresponder

- **`glTexImage2D()`** designates entire texture image
- Call **`glTexSubImage2D()`** to specify an offset and width, height parameters to use only part of texture image

Corresponder function

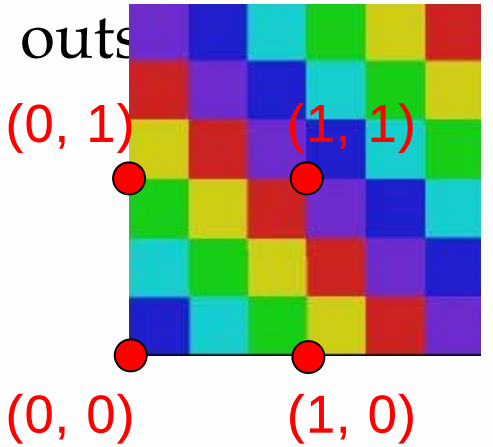
- Maps from (u, v) to texture image coordinates (s, t) in units of pixels
- (u, v) in range [0, 1] are displayed; corresponder decides:
  - What part of texture image this references (all or only part)
  - Wrapping: How to handle values outs [0, 1] (including negative)



for example, a 256 x 256 texture image

Corresponder function

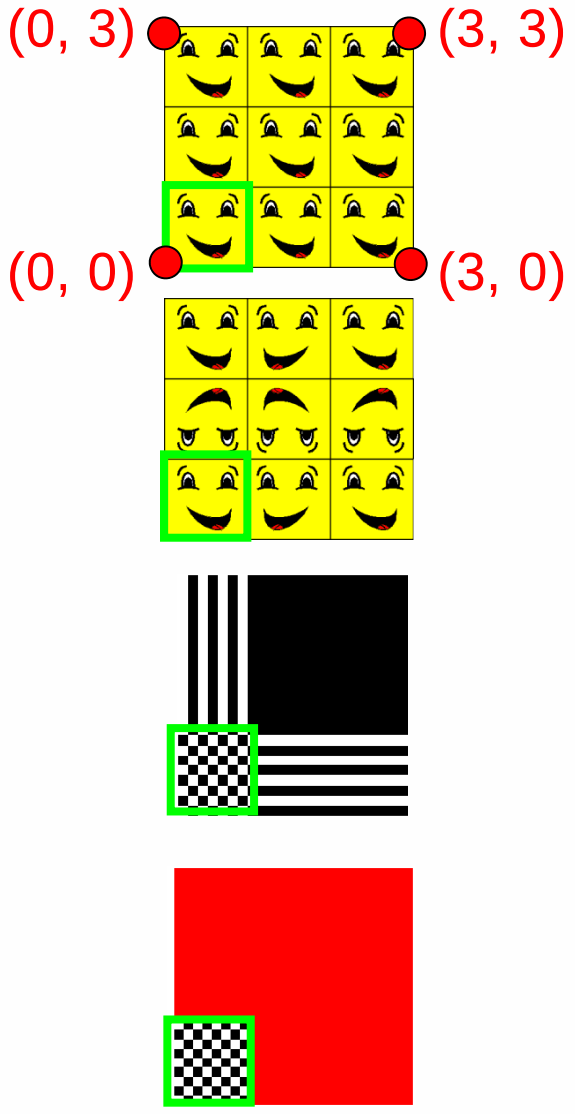
- Maps from (u, v) to texture image coordinates (s, t) in units of pixels
- (u, v) in range [0, 1] are displayed; corresponder decides:
  - What part of texture image this references (all or only part)
  - Wrapping: How to handle values outs [0, 1] including negative



for example, a 256 x 256 texture image

Wrapping modes

- **repeat**: Start entire texture over
- **mirror**: Flip copy of texture in each direction
  - Get continuity of pattern
- **clamp to edge**: Extend texture edge pixels
- **clamp to border**: Surround with border color



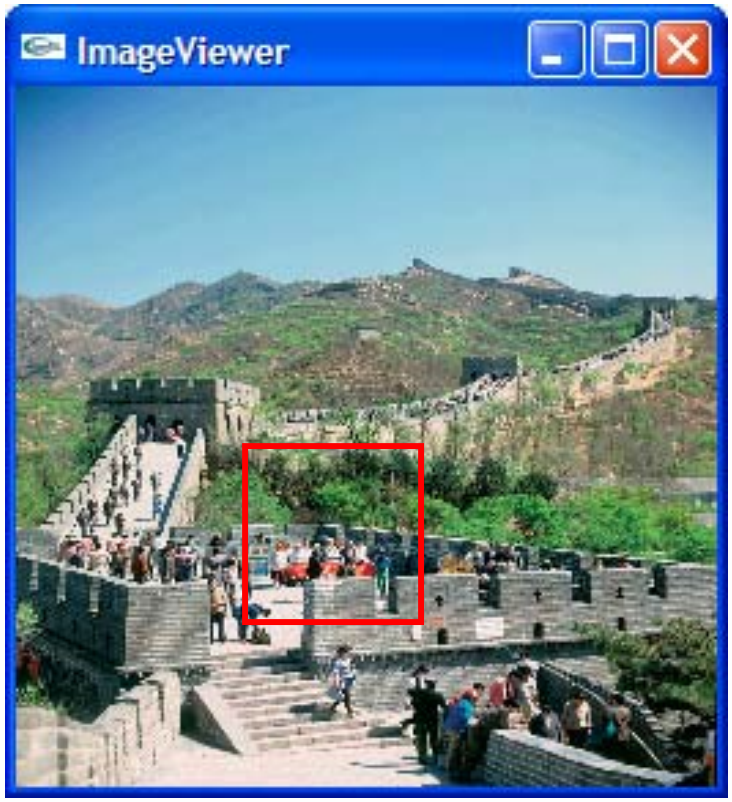
courtesy of Microsoft

OpenGL: Border handling

- Set with **`glTexParameter(GL_TEXTURE_2D, pname, params)`**, where:
  - **`pname`**
    - **`GL_TEXTURE_WRAP_S`**, **`GL_TEXTURE_WRAP_T`**
    - **`GL_TEXTURE_BORDER_COLOR`**
  - **`params`**
    - **`GL_CLAMP`**
    - **`GL_CLAMP_TO_EDGE`**
    - **`GL_REPEAT`**
    - **`[r, g, b, alpha]`**
- Default is to repeat; default border color in **`GL_CLAMP`** mode is **`(0, 0, 0, 0)`**

How can we see more?

- Our world view only allows us to see so much
- We need to define a mapping from our desired world view to our screen.

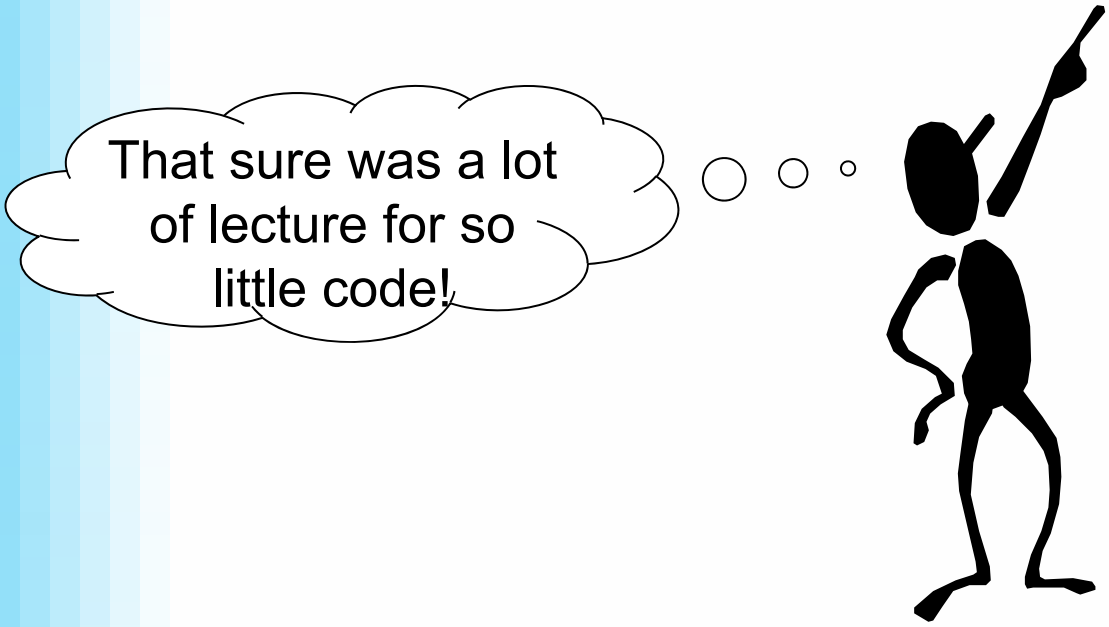


Speaking of Mouse Click

```
void onMouseButton(int button, int state, int x, int y) {
    double mouseX, mouseY;
    double dx, dy;

    if ((button == GLUT_LEFT_BUTTON) && (state == GLUT_DOWN)){
        mouseX = xScreenToWorld(x);
        mouseY = yScreenToWorld(y);
        dx = (world.r - world.l)/4;
        dy = (world.t - world.b)/4;
        world.l = mouseX - dx;
        world.r = mouseX + dx;
        world.b = mouseY - dy;
        world.t = mouseY + dy;
        initialize();
        display();
    } else
        if ((button == GLUT_RIGHT_BUTTON) && (state == GLUT_DOWN))
        {   mouseX = xScreenToWorld(x);
            mouseY = yScreenToWorld(y);
            dx = (world.r - world.l);
            dy = (world.t - world.b);
            world.l = mouseX - dx;
            world.r = mouseX + dx;
            world.b = mouseY - dy;
            world.t = mouseY + dy;
            initialize();
            display();
        }
}
```

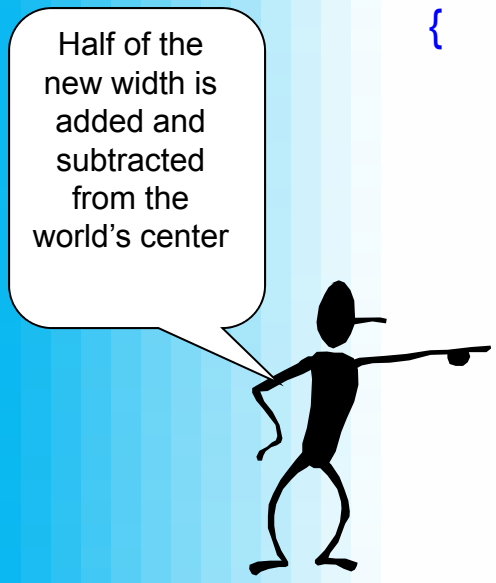
Screen-to-World Mapping

```
double xScreenToWorld(int x) {
    return ((world.r - world.l) * (x - viewport[0]) / viewport[2]) + world.l;
}
```



Speaking of Resize

- Resize gets called when the window size changes




```
void resize(int w, int h)
{
        double l, r;
        glViewport(0, 0, (GLsizei) w, (GLsizei) h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        l = 0.5*(world.r + world.l) - (0.5*(world.t - world.b)*w)/h;
        r = 0.5*(world.r + world.l) + (0.5*(world.t - world.b)*w)/h;
        world.l = l;
        world.r = r;
        gluOrtho2D(world.l, world.r, world.b, world.t);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glGetIntegerv(GL_VIEWPORT, viewport);
}
```

Why didn't he show us yScreenToWorld()?

1) He's lazy, and was running out time preparing the lecture
2) There must be some trick
3) He's probably going to put something like it on a quiz at some point
4) All of the above

How does the world's width change on a left mouse click?

- Hint

```
dx = (world.r - world.l)/4;
dy = (world.t - world.b)/4;
world.l = mouseX - dx;
world.r = mouseX + dx;
world.b = mouseY - dy;
world.t = mouseY + dy;
```

1) It's a quarter of original window's size
2) It's half of original window's size
3) It's unrelated to the original windows size
4) This code will never work

Next Time

- 2D Graphics and Imaging in OpenGL