3D Graphics Programming Tools

OpenGL Coordinates and Viewing

Dr. Xianhui Cherry Che

x.che@qmul.ac.uk

Learning Objectives

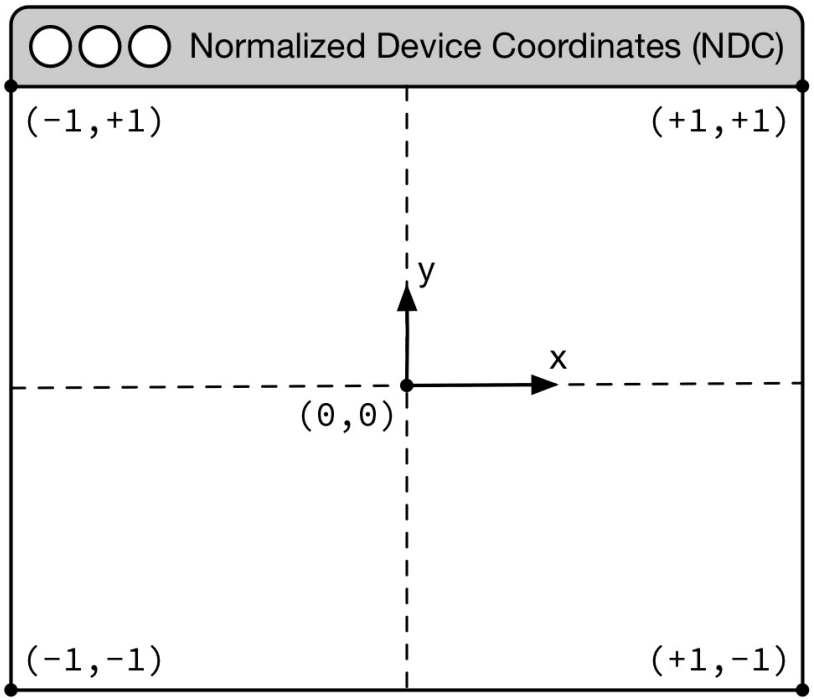
- Understand the OpenGL coordinate spaces from different perspectives and how they can be applied in the development process
- Understand the concept of viewing and the practice of projection in OpenGL
- Practise clipping in OpenGL

Topics

- Coordinate Spaces
- Concept of OpenGL Viewing
- OpenGL Projection
- Viewport Clipping

Recap: Normalized Device Coordinates

- Normalized Device Coordinates (NDC) is a display coordinate system that is **screen-independent**; it encompasses a 2D or 3D positions where the x, y, and/or z components range from -1 to 1.

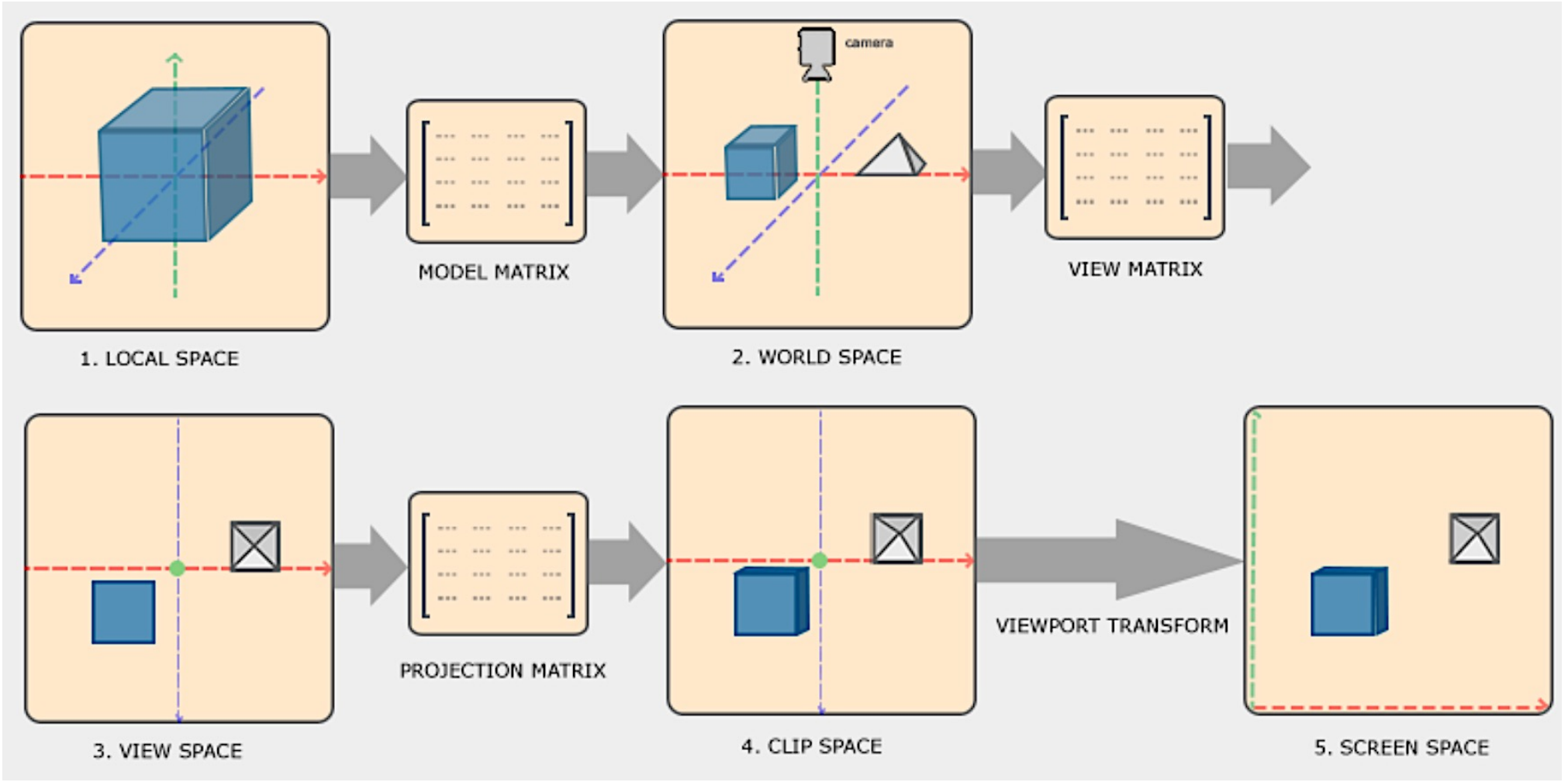


Five Coordinate Spaces

There are a total of 5 different coordinate spaces that are of importance to us:

- **Local space (or Object space)**
- **World space**
- **View space (or Eye space)**
- **Clip space**
- **Screen space**

The Global Picture



The Global Picture

1. **Local Space:** Local coordinates are the coordinates of your object relative to its local origin; they are the coordinates the object begins in.
2. **World Space:** The next step is to transform the local coordinates to world-space coordinates which are coordinates in respect of a larger world. These coordinates are relative to some global origin of the world, together with many other objects also placed relative to this world's origin.

The Global Picture

3. **View Space:** Next, the world coordinates are transformed to view-space coordinates in such a way that each coordinate is as seen from the camera or viewer's point of view.
4. **Clip Space:** The coordinates in view space are then projected to clip coordinates. Clip coordinates are processed to the -1.0 and 1.0 range and determine which vertices will end up on the screen. Projection to clip-space coordinates can add perspective if perspective projection is used.
5. **Screen Space:** Lastly, the clip coordinates are projected to screen coordinates using **viewport** transformation, which transforms the coordinates from -1.0 and 1.0 to the coordinate range defined by **`glViewport`**. The resulting coordinates are then sent to the rasterizer to turn them into fragments.

What is the Rationale for Having Different Spaces?

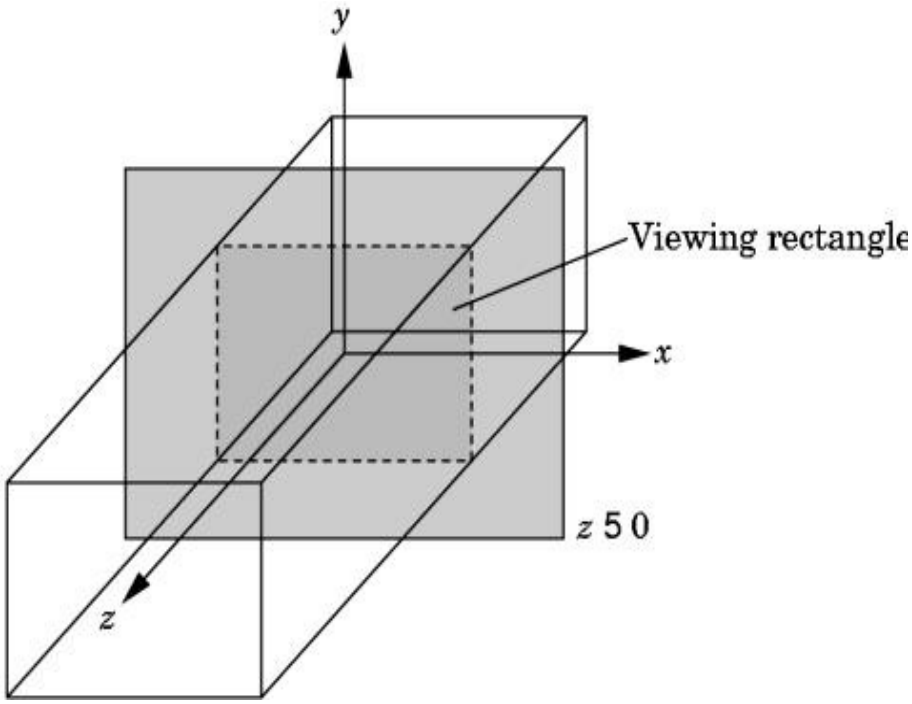
- The reason to transform vertices into all these different spaces is that some operations make more sense or are easier to use in certain coordinate systems. For example:
  - When modifying an object, it makes most sense to do this in local space
  - While calculating certain operations on the object with respect to the position of other objects, it makes most sense to do so in world coordinates.
- Although it is possible to define one transformation matrix that goes from local space to clip space all in one go, it offers less flexibility.

Topics

- Coordinate Spaces
- Concept of OpenGL Viewing
- OpenGL Projection
- Viewport Clipping

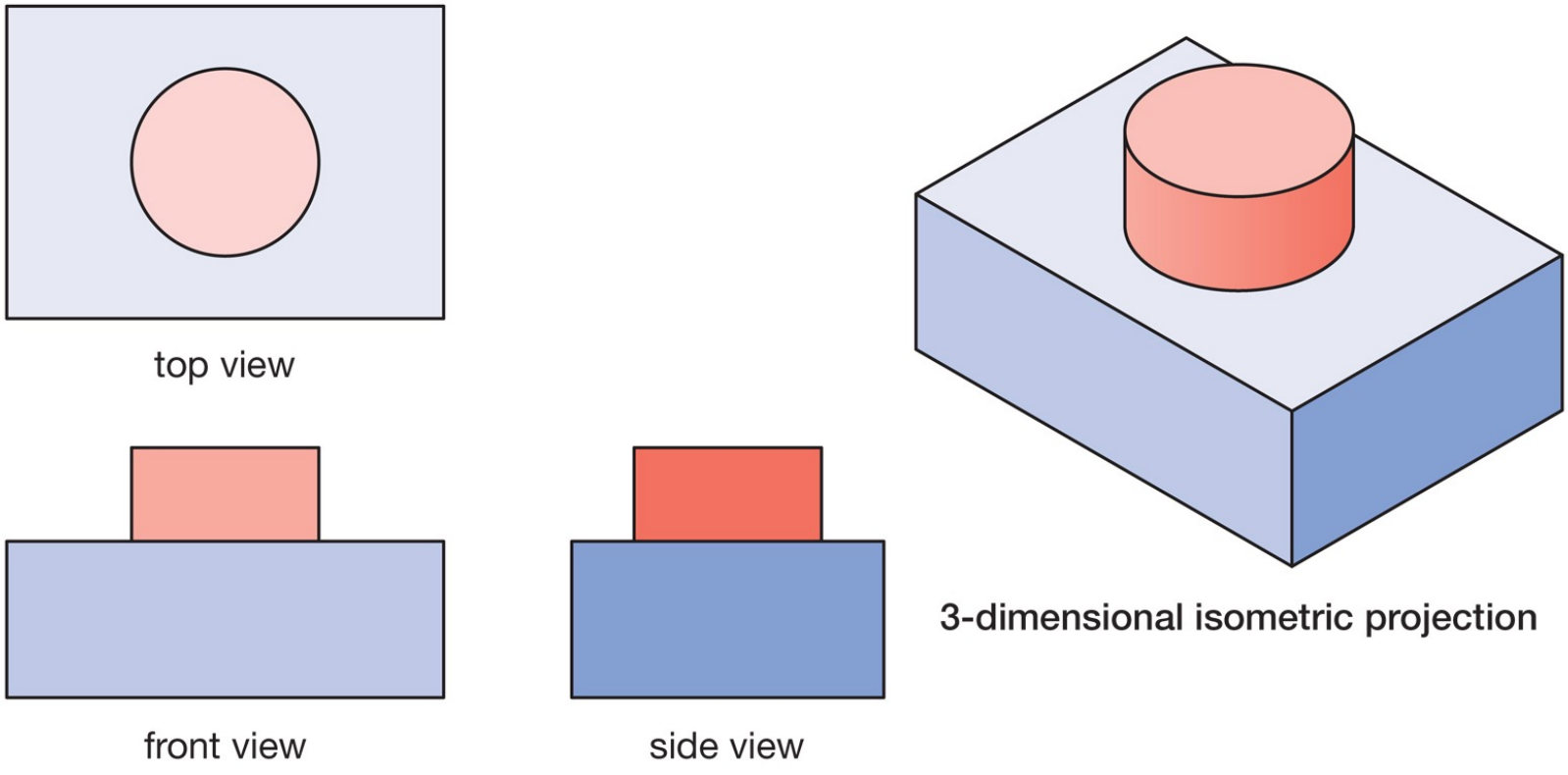
Viewing in Coordinate Systems

- **Viewing volume** determines what appears in an image.
- It is essentially the part of the world that is visible in the image (i.e. what can be seen).
- It is defined by viewing specifications.



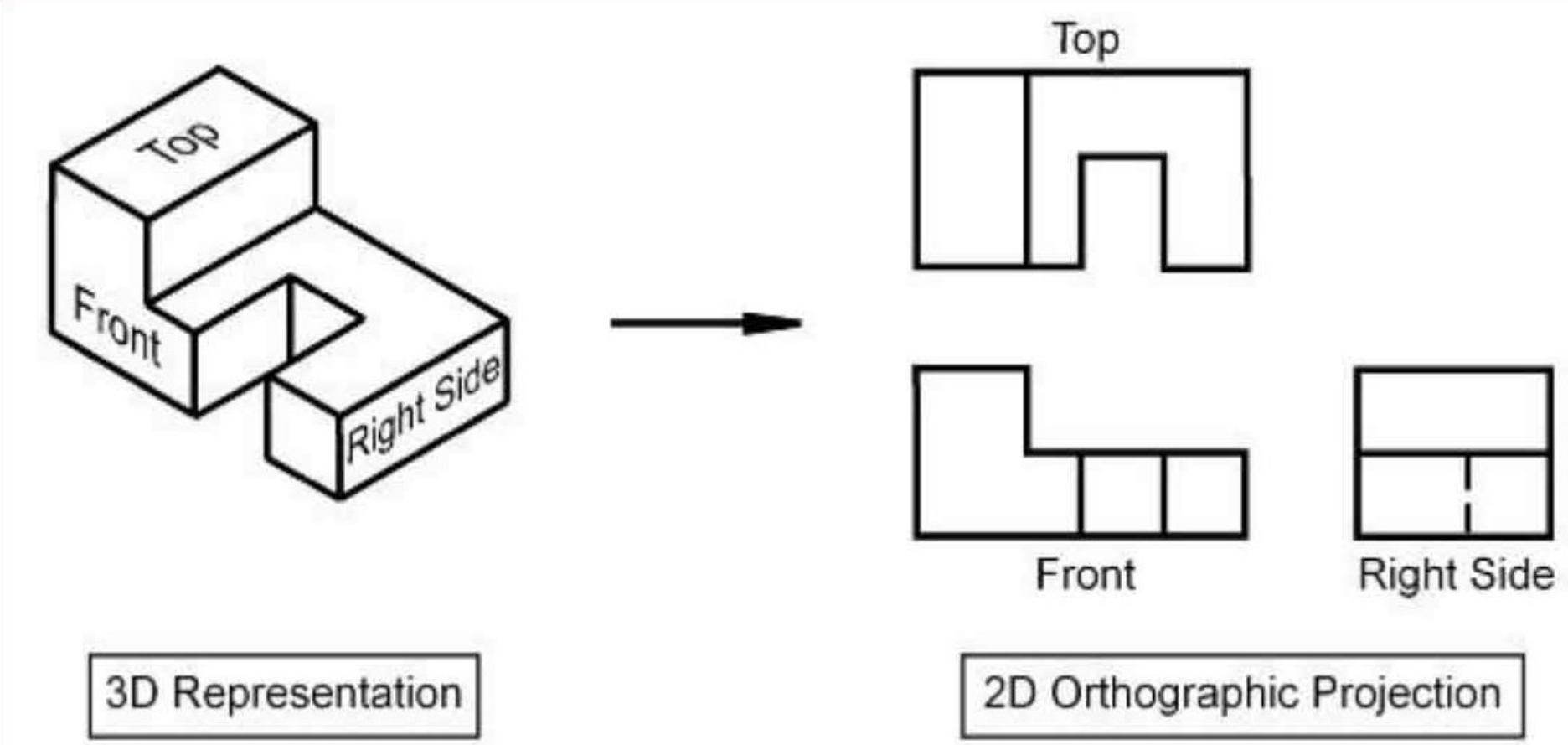
Orthographic Viewing

- Example 1



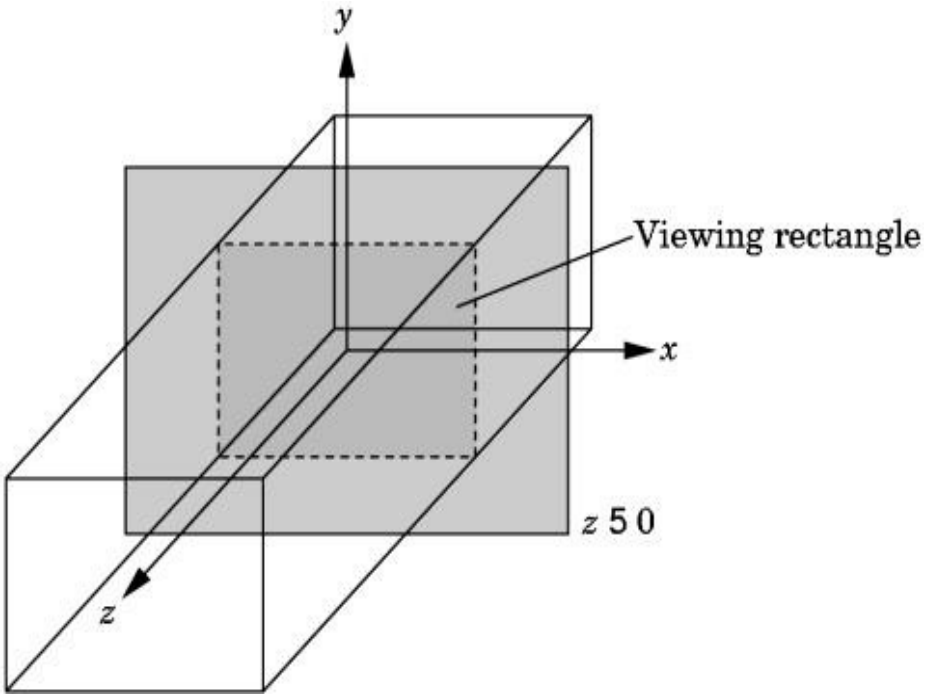
Orthographic Viewing

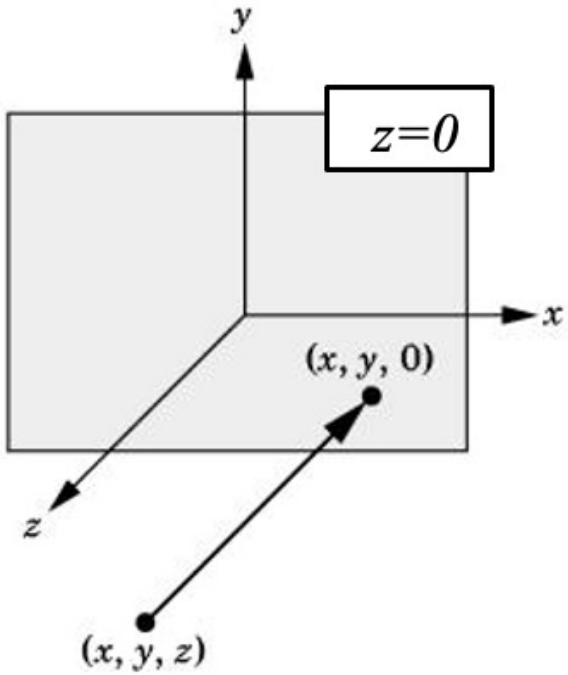
- Example 2



Orthographic Viewing in OpenGL

- In the default orthographic view, points are projected forward along the z axis onto the plane $z=0$



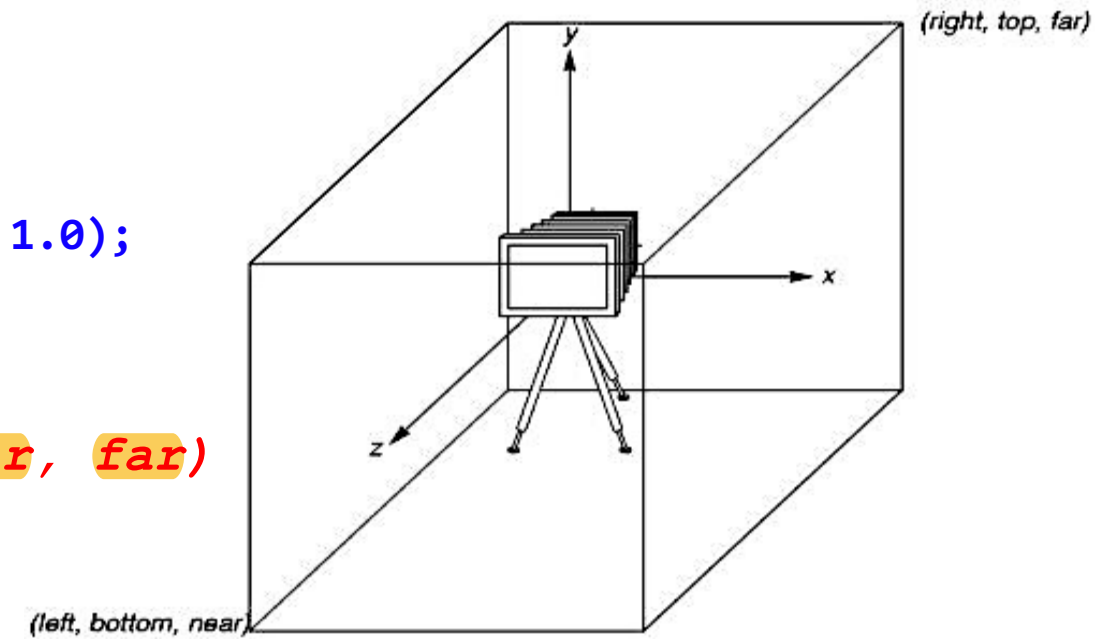


OpenGL Camera

- OpenGL places a default camera at the origin of the space pointing in the negative z direction
- The default viewing volume is a box centered at the origin with a side of length 2

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

(left, right, bottom, top, near, far)



GL Functions

- `glMatrixMode` sets the current matrix mode
- `glLoadIdentity` replaces the current matrix with the identity matrix
- `glOrtho` describes a transformation that produces a parallel projection

Transformations and Viewing

- In OpenGL, projection is carried out by a projection matrix (transformation)
- Set the matrix mode first

```
glMatrixMode (GL_PROJECTION);
```

- Start with an identity matrix (the default setting)

```
glLoadIdentity();
```

- Alter the matrix with a projection matrix that gives the view volume

```
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

2D and 3D Viewing

- In 3D viewing:

```
glOrtho(left, right, bottom, top, near, far)
```

- Two-dimensional vertex commands place all vertices in the plane z=0
- Function `gluOrtho2D` can be used in 2D applications instead:

```
gluOrtho2D(left, right, bottom, top)
```

- In two dimensions, the view volume becomes a **view rectangle**.

Topics

- Coordinate Spaces
- Concept of OpenGL Viewing
- OpenGL Projection
- Viewport Clipping

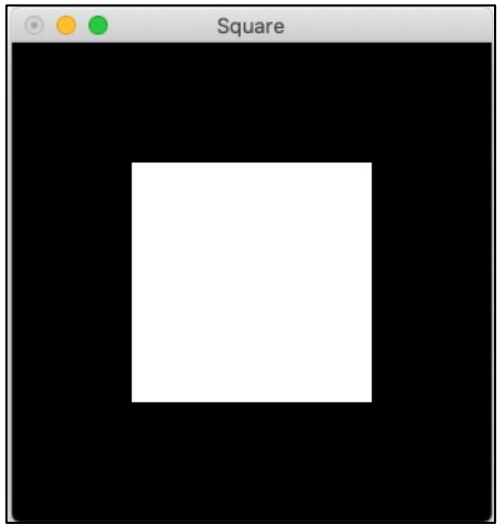
Draw a Square – 3D Projection View

```
#include <GLUT/glut.h>
void drawSquare(){
    glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}


void init () {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
    }
```

```
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutInitWindowPosition(-1,-1);
    glutCreateWindow("Square");
    glutDisplayFunc(drawSquare);
    init();
    glutMainLoop();
}
```

Output:



Not much difference so far

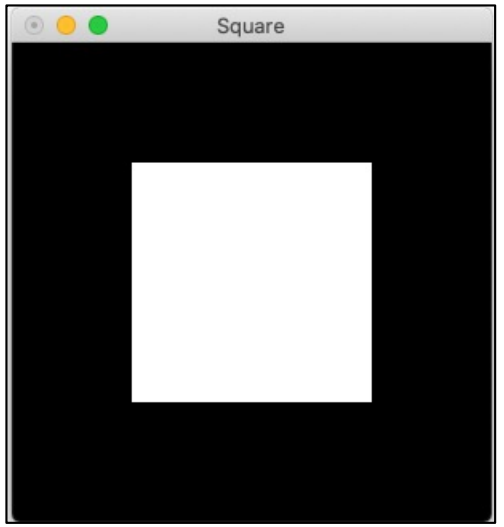
Draw a Square – 2D Projection View

```
#include <GLUT/glut.h>
void drawSquare(){
    glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}


void init () {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
    }
```

```
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutInitWindowPosition(-1,-1);
    glutCreateWindow("Square");
    glutDisplayFunc(drawSquare);
    init();
    glutMainLoop();
}
```

Output:



Not much difference so far

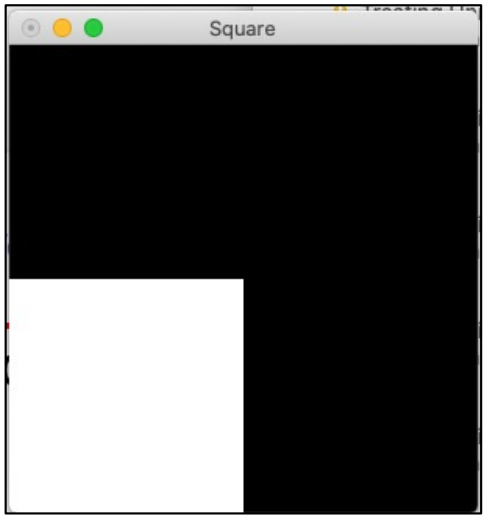
Draw a Square – 2D Projection View

```
#include <GLUT/glut.h>
void drawSquare(){
    glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}


void init () {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D(-0.5, 1.5, -0.5, 1.5);
    }
```

```
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutInitWindowPosition(-1,-1);
    glutCreateWindow("Square");
    glutDisplayFunc(drawSquare);
    init();
    glutMainLoop();
}
```

Output:

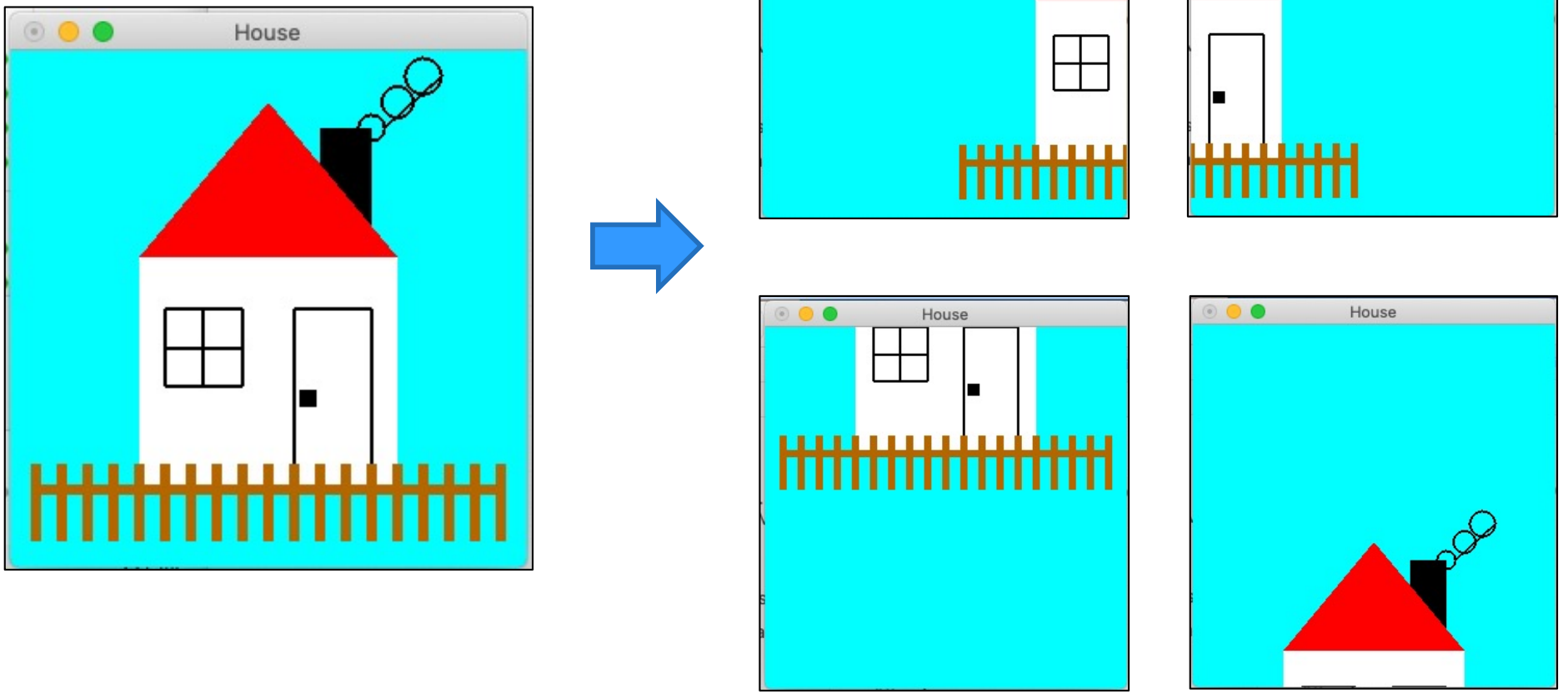


The view has changed!

GLUT Functions

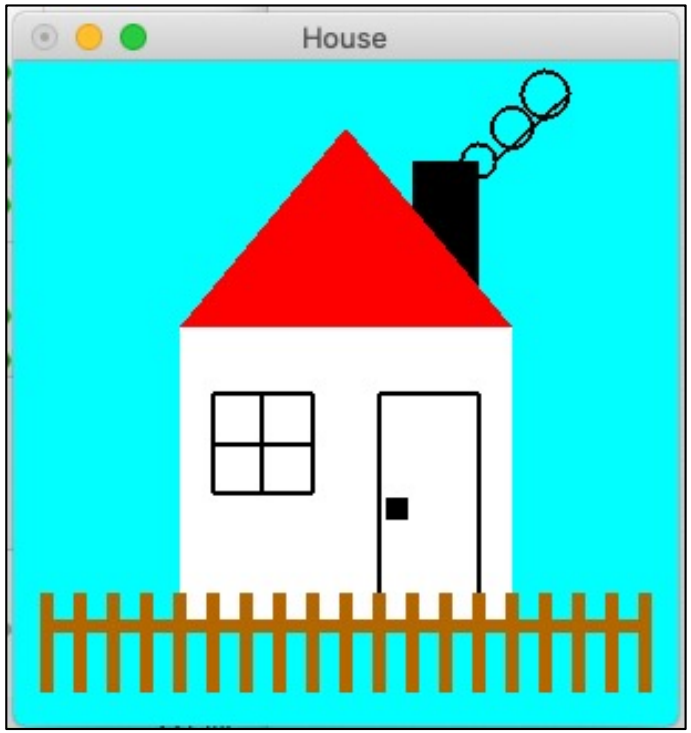
- `glutInitDisplayMode` sets the initial display mode

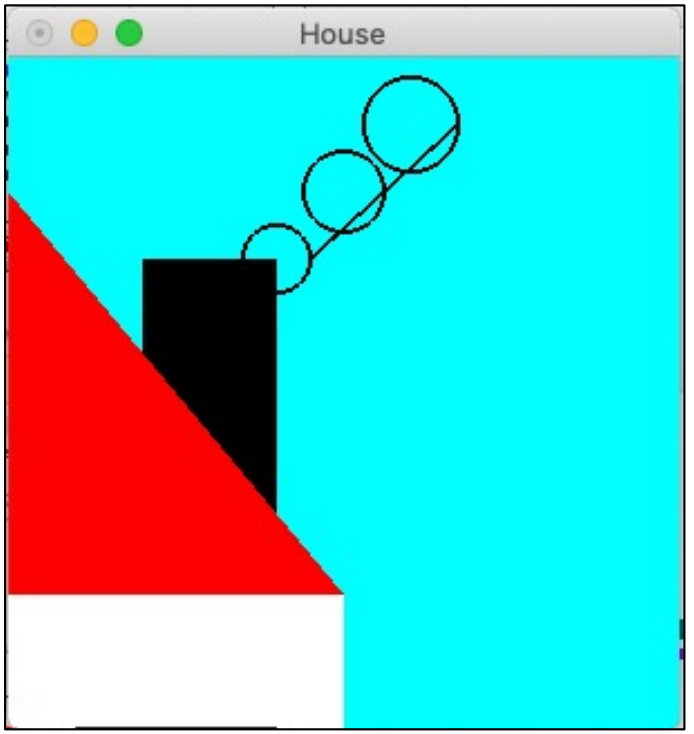
To Do in Tutorial 1



Zoomed Image

- How to obtain a zoomed image using projection?

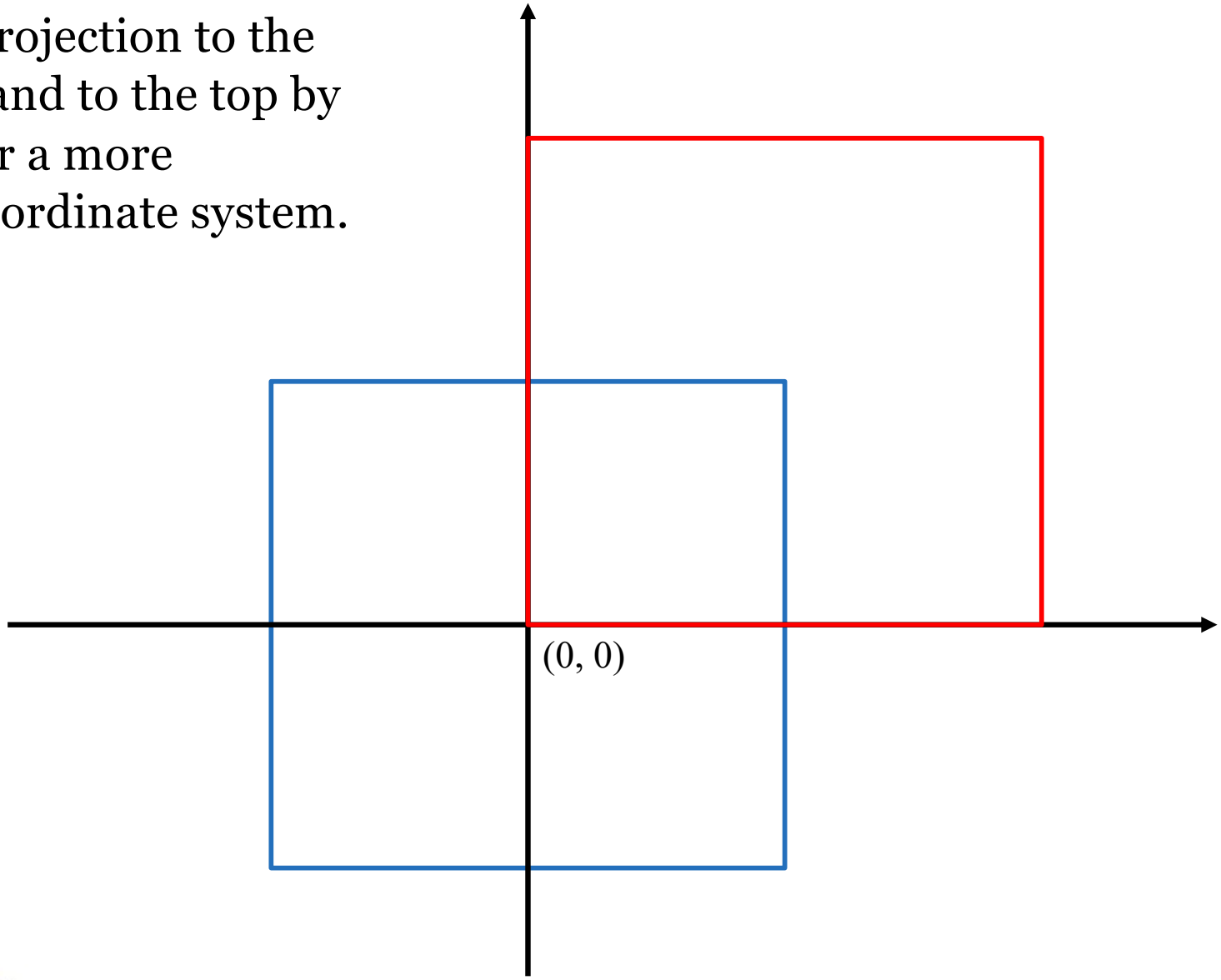





```
gluOrtho2D(0, 1, 0, 1);
```

Using Traditional Coordinates

Moving the projection to the right by half and to the top by half allows for a more traditional coordinate system.



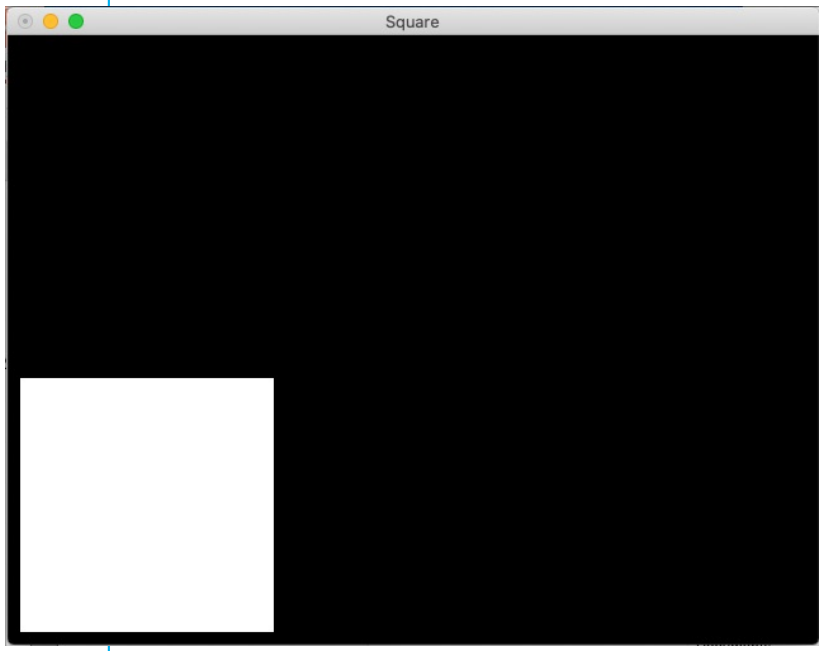
Using Traditional Coordinates

```
#include <GLUT/glut.h>
void init(void) {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glColor3f(1.0, 1.0, 1.0);
   glPointSize(5.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0, 640, 0, 480);
}

void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(10, 10);
        glVertex2f(210, 10);
        glVertex2f(210, 210);
        glVertex2f(10, 210);
        glEnd();
        glFlush();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Square");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```

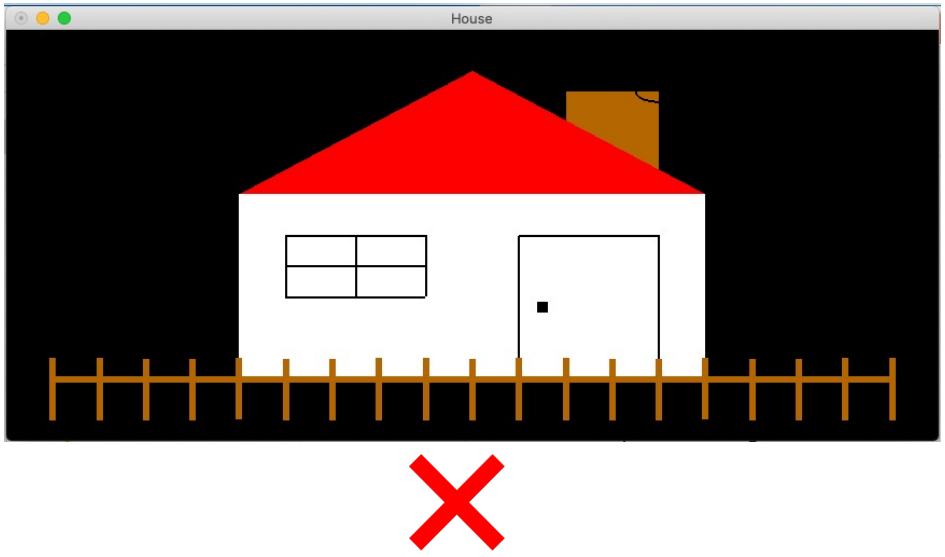
Can also use glVertex2i()

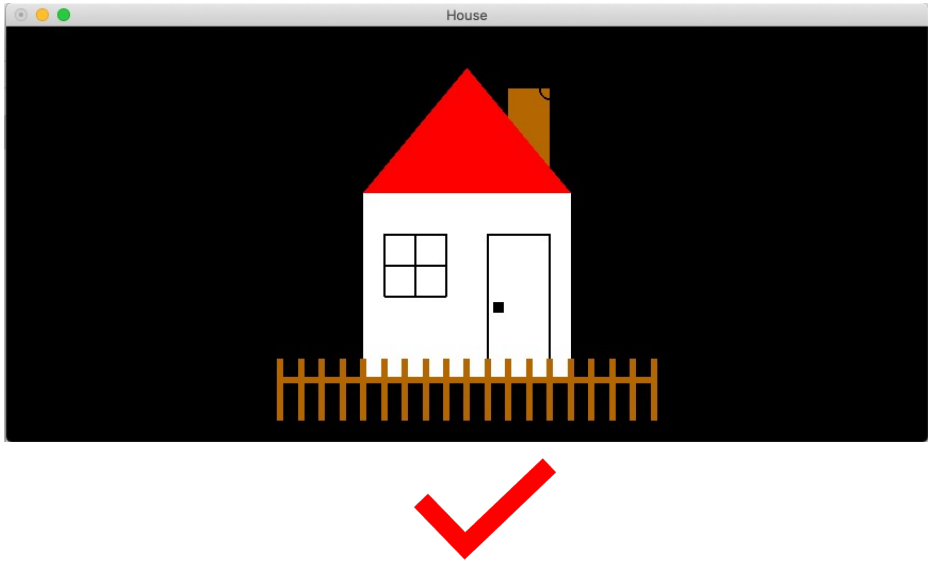
Output:



Reshape an Image

- A `reshape()` function can be used to ensure consistent aspect ratio between clipping-area and viewport. The graphics sub-system passes the window's width and height, in pixels, into the `reshape()`.





```
#include <GLUT/glut.h>
#include <math.h>
void drawHouse(){… … }
void display() {
    drawHouse();
    glFlush();}

void init() { … … }

void reshape(GLsizei width, GLsizei height) {
   if (height == 0) height = 1;
   GLfloat aspect = (GLfloat)width / (GLfloat)height;
   glLoadIdentity();
   if (width >= height) {
      gluOrtho2D(-1.0 * aspect, 1.0 * aspect, -1.0, 1.0);
   } else {
     gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);
   }}

int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutCreateWindow("House");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    init();
    glutMainLoop();
}
```

Reshape an Image

- The `reshape()` function with comments:

```
void reshape(GLsizei width, GLsizei height) {// GLsizei for non-negative integer

   // Compute aspect ratio of the new window
   if (height == 0) height = 1;                // To prevent divide by 0
   GLfloat aspect = (GLfloat)width / (GLfloat)height;

   // Change the aspect ratio
   glLoadIdentity();             // Reset the projection matrix
   if (width >= height) {
     // aspect >= 1, set the height from -1 to 1, with larger width
      gluOrtho2D(-1.0 * aspect, 1.0 * aspect, -1.0, 1.0);
   } else {
      // aspect < 1, set the width to -1 to 1, with larger height
     gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);
   }
}
```

What has Happened?

- First, `main` defines a reshape callback:

```
glutReshapeFunc(reshape);
```

  - **`glutReshapeFunc`** is similar to **`glutDisplayFunc`** in that it sets up a callback - this time, the callback is the function to call when the window is reshaped (i.e. resized).
- The `reshape` function defines what to do when the window is resized.
  - Determines the aspect ratio. Reset the projection matrix.
  - If the aspect ratio is not 1, then project the image accordingly to maintain the image ratio.

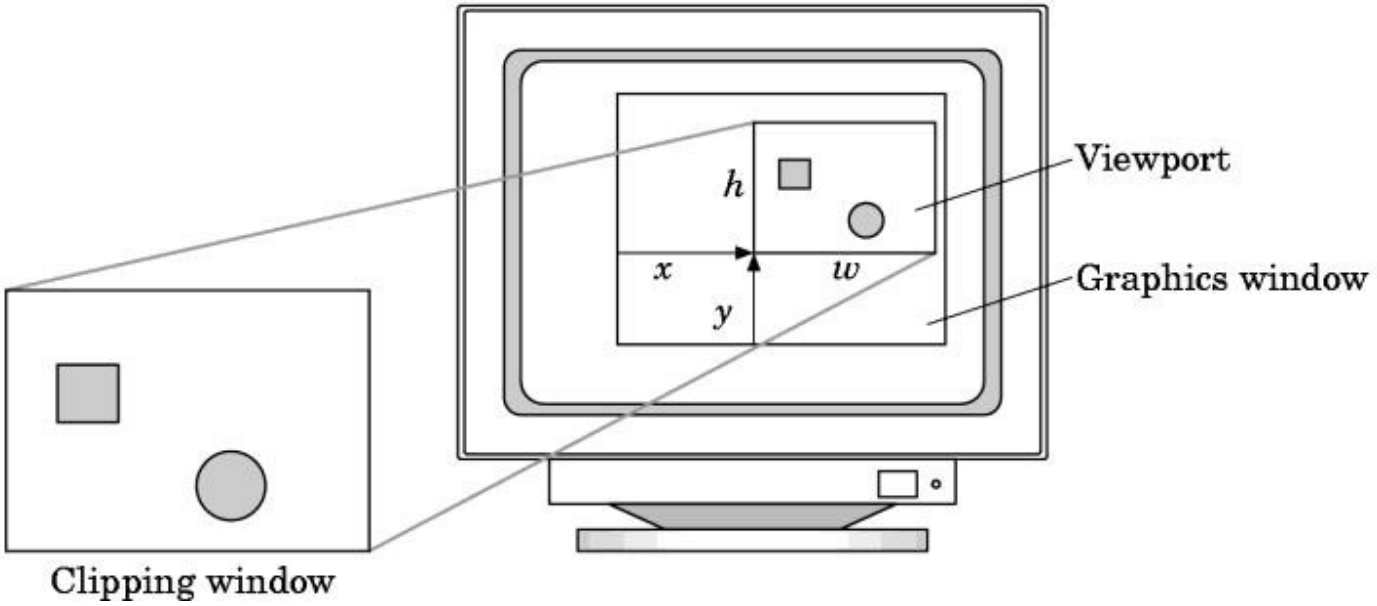
Topics

- Coordinate Spaces
- Concept of OpenGL Viewing
- OpenGL Projection
- Viewport Clipping

Viewports

- `glViewport` specifies the transformation of x and y from NDC to window coordinates.

```
glViewport(x,y,w,h)
```

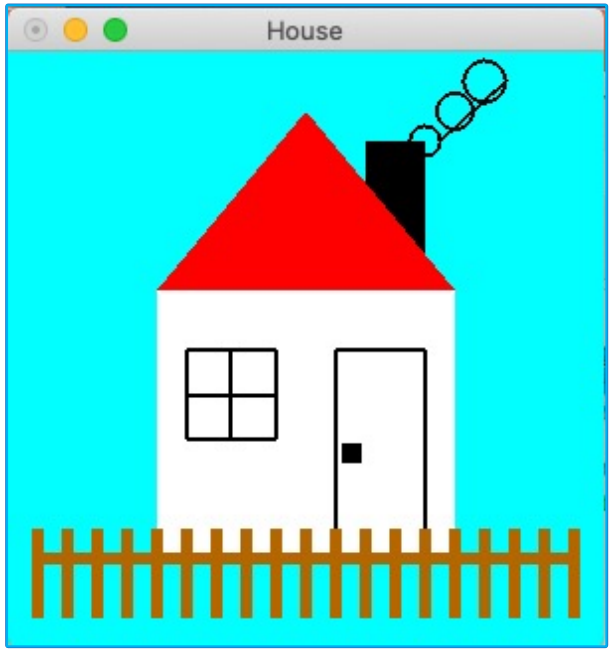


Viewport Example 1

```
#include <GLUT/glut.h>
#include <math.h>
void drawHouse(){ … … }
void init() {
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f (0.0, 0.0, 0.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D(-1, 1, -1, 1);     }

void display() {
    glViewport(0, 0, 300, 300);
    drawHouse();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutCreateWindow("House");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```

Output:



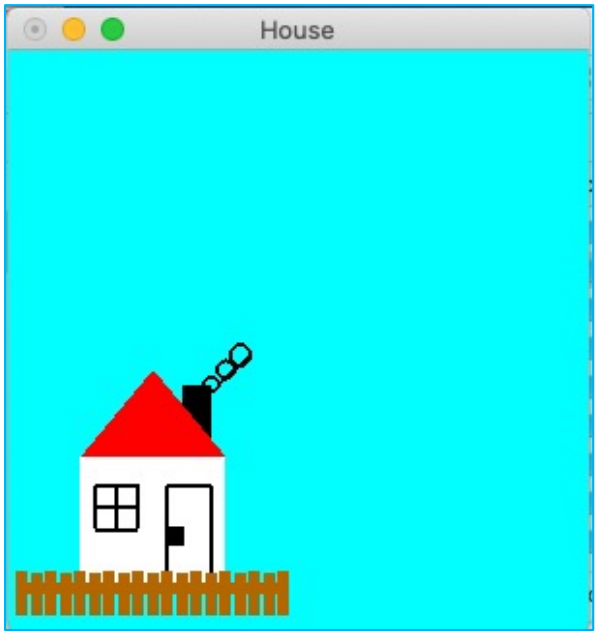
Not much difference so far

Viewport Example 2

```
#include <GLUT/glut.h>
#include <math.h>
void drawHouse(){ … … }
void init() {
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f (0.0, 0.0, 0.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D(-1, 1, -1, 1);     }

void display() {
    glViewport(0, 0, 150, 150);
    drawHouse();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutCreateWindow("House");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```

Output:



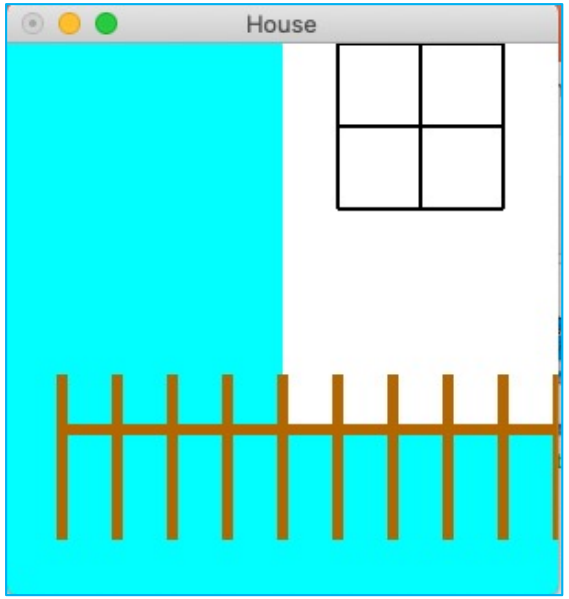
Resize the viewport.
Zoomed out!

Viewport Example 3

```
#include <GLUT/glut.h>
#include <math.h>
void drawHouse(){ … … }
void init() {
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f (0.0, 0.0, 0.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D(-1, 1, -1, 1);     }

void display() {
    glViewport(0, 0, 600, 600);
    drawHouse();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutCreateWindow("House");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```

Output:



Resize the viewport again.
Zoomed in!

What has Happened?

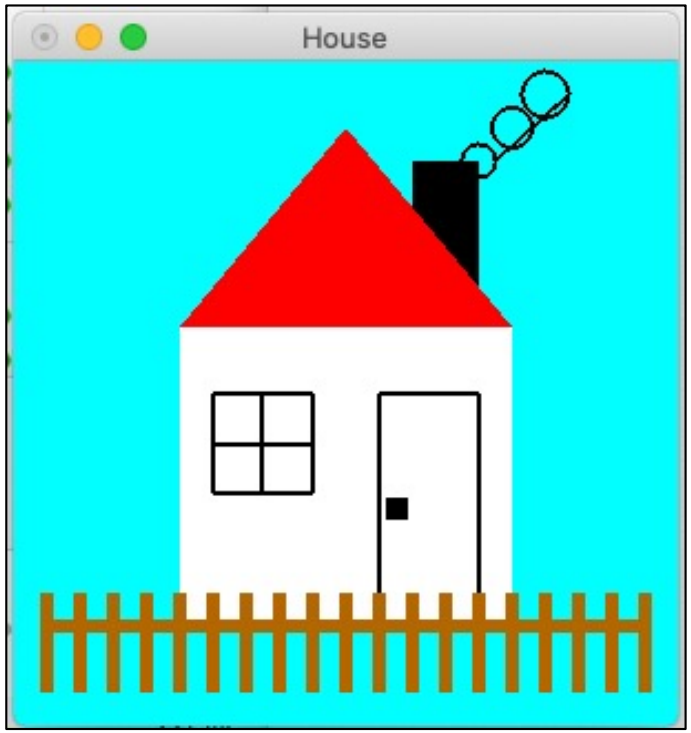
- **`glViewport`** defines the lower left corner and dimensions of the drawing window:

```
void glViewport ( GLint x, GLint y, GLsizei width, GLsizei height );
```

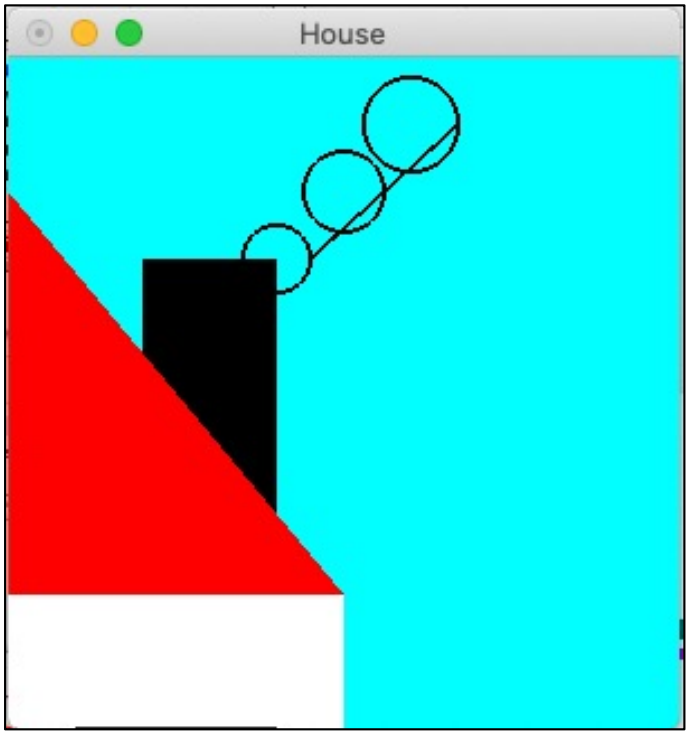
- GLint and GLsizei are special OpenGL types; you can just use integer values. These are pixel coordinates in the drawing window.

One More Example

- How to obtain a zoomed image using viewport?

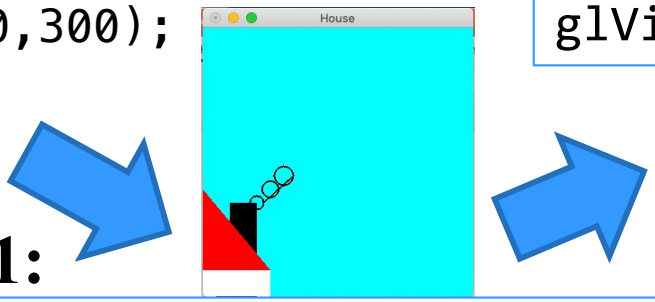


glutInitWindowSize(300,300);



Step2:

```
glViewport(0, 0, 600, 600);
```

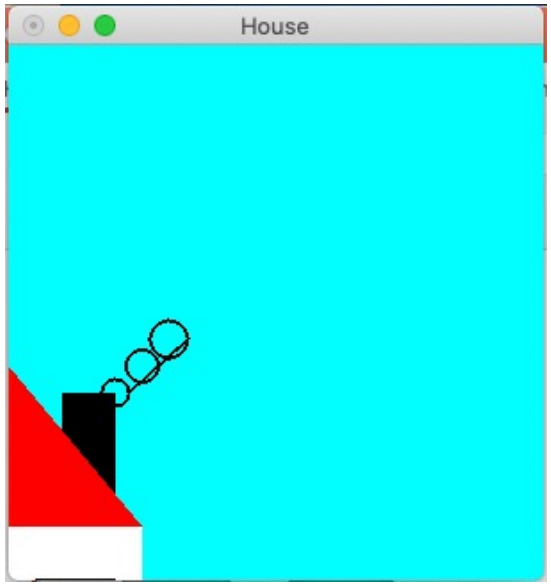


Step1:

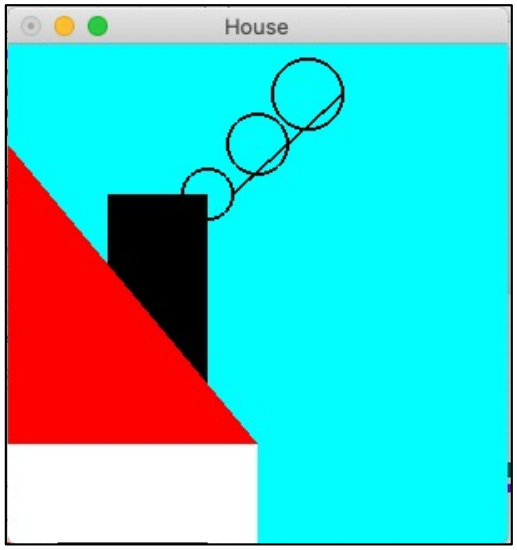
```
gluOrtho2D(0, 2, 0, 2);
```

Queen Mary University of London

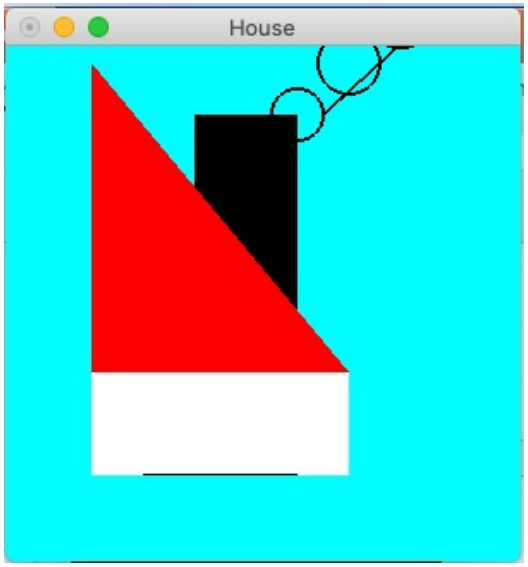
From Clipped Space to Screen Space



gluOrtho2D(0, 2, 0, 2);



glViewport(0, 0, 600, 600);



glViewport(50, 50, 600, 600);

Tiles Images

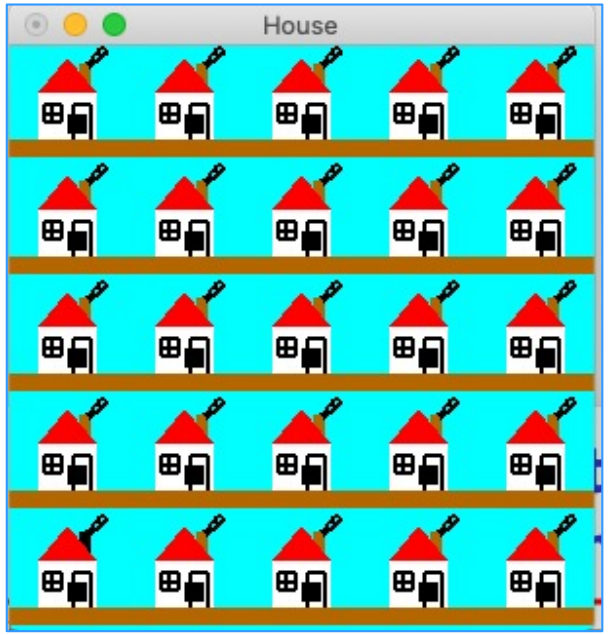
- How to use viewport to tile a screen?

Tiles Images

```
#include <GLUT/glut.h>
#include <math.h>
void drawHouse(){… …}

void display() {
    int i, j;
    for (i=0; i<5; i++)
        for (j=0; j<5; j++) {
            glViewport(i*60, j*60, 60, 60);
            drawHouse();
        }
    glFlush();
}
void init() {
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f (0.0, 0.0, 0.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D(-1, 1, -1, 1);
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutCreateWindow("House");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```

Output:



Questions?

x.che@qmul.ac.uk