2D Graphics

Lecture 4
Section 2.7

Robb T. Koether

Hampden-Sydney College

Wed, Aug 31, 2011

Outline

3D Graphics

- In three-dimensional graphics, objects are drawn in 3D space.
- As they are drawn, they are projected onto a plane and then written to the frame buffer.
- Unless we instruct the GPU otherwise, each object overwrites previous objects in the same location in the frame buffer.
- To avoid that, we turn depth testing on and specify the desired test.
- Then we clear the depth buffer before drawing the scene.

Setting up the Depth Buffer

```
void init()
{
    glClearColor(0.8, 0.8, 0.8, 0.0);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glClearDepth(1.0);
        ⋮
    return;
}
```

2D Graphics

- In two-dimensional graphics, objects are first drawn as 3D objects.
- Then
  - The z-coordinate is ignored (normally $z = 0$).
  - Objects are projected orthographically onto the xy-plane.
  - Depth testing is turned off.
  - The last object drawn appears on top.

2D Graphics

- Since there is no perspective viewing, we need only keep track of the *x* and *y* world coordinates at the edges of the viewport.
- We will call these `xmin`, `xmax`, `ymin`, and `ymax`.
- The function call will be

```
gluOrtho2D(xmin, xmax, ymin, ymax);
```

The Reshape Function

Example (The Reshape Function)

```
void reshape(int w, int h)
{
    if (w > 0 && h > 0)
    {
        xmin = ...    // Calculate new bounds
        xmax = ...    // if necessary
        ymin = ...
        ymax = ...
        screenWidth = w;
        screenHeight = h;
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(xmin, xmax, ymin, ymax);
        glViewport(0, 0, w, h);
    }
    return;
}
```

The Viewport

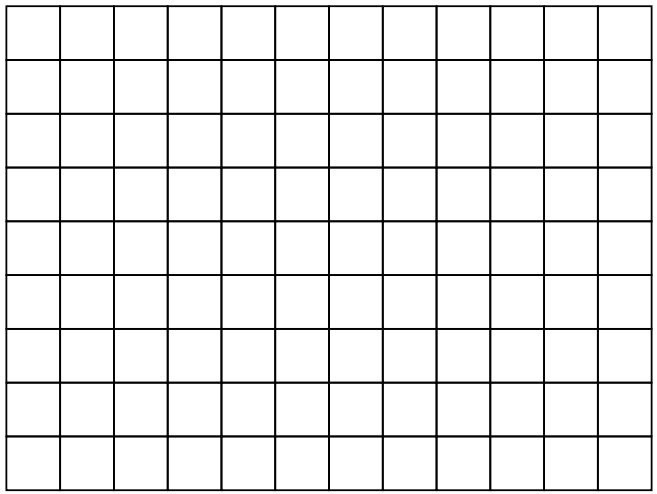
- The viewport is the part of the window in which the scene is displayed.
- Normally, this is the entire window.
- Typically, we initialize the window size to 640×480.
- The function call

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

  establishes the viewport on the screen.

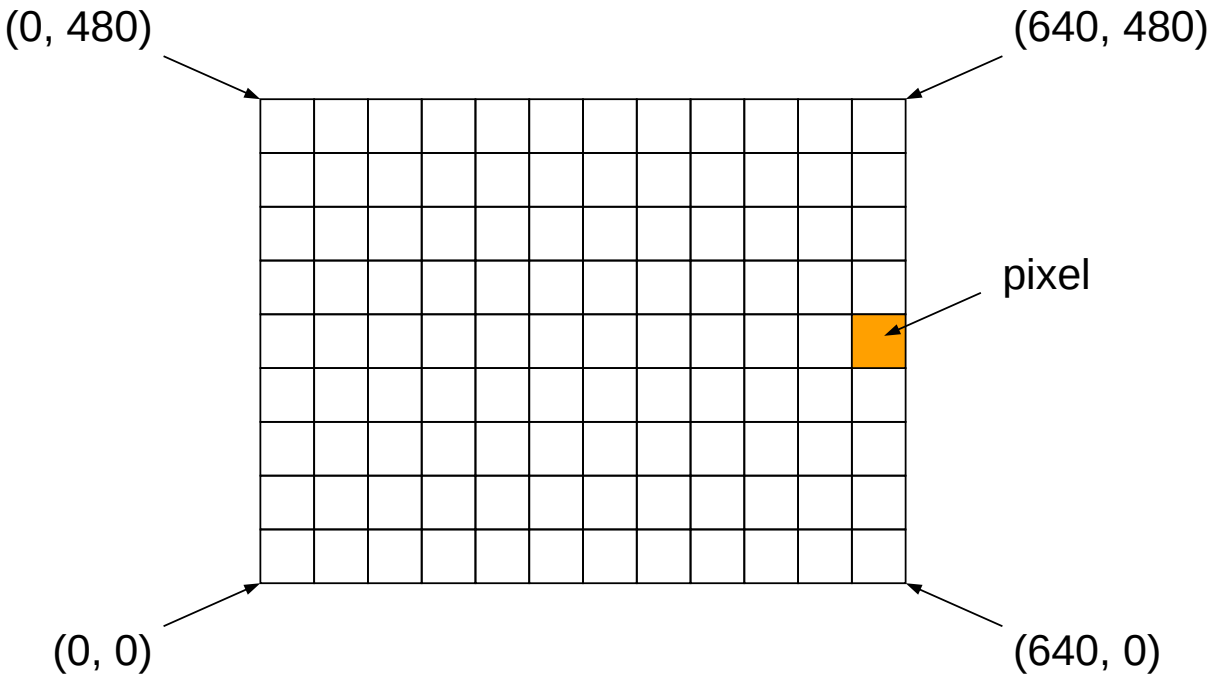
Viewport Coordinates

- The coordinates are based on the “gridlines” between the pixels.



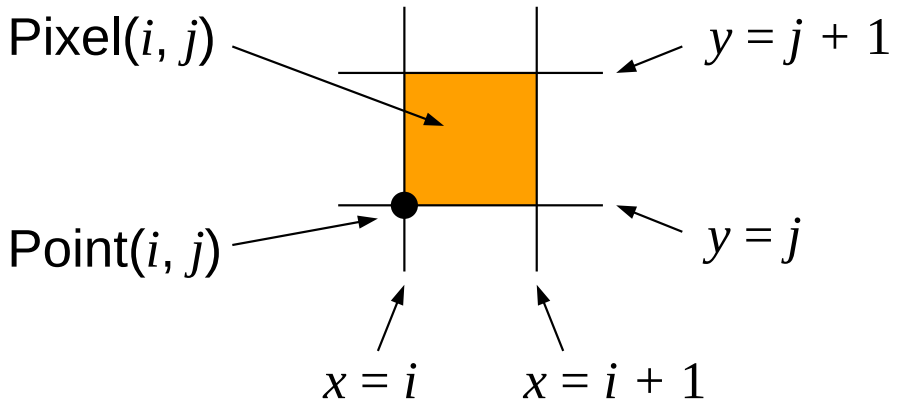
Viewport Coordinates

- The coordinates are based on the “gridlines” between the pixels.



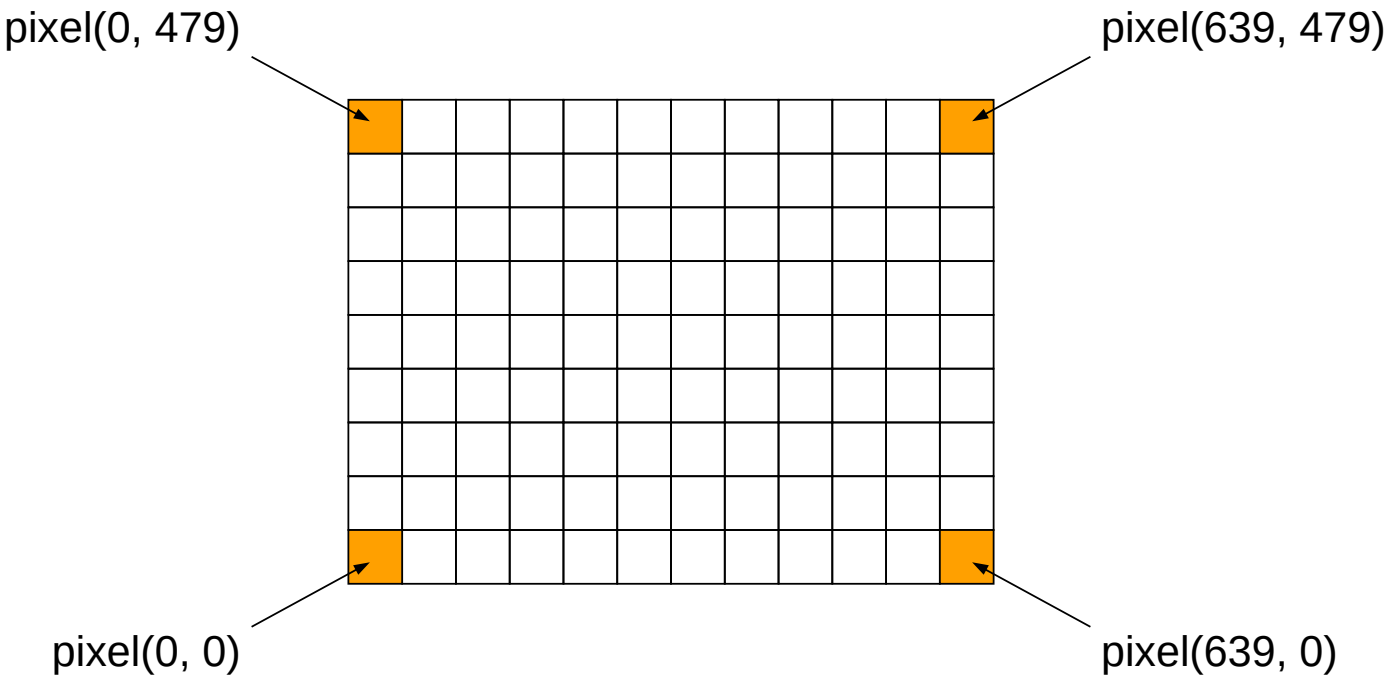
Viewport Coordinates

- Pixel (i, j) is located between gridlines $x = i$ and $x = i + 1$, and $y = j$ and $y = j + 1$.



Viewport Coordinates

- The corner pixels.



World Coordinates

- To make the two-dimensional world coordinates identical with the viewport coordinates, write

```
xmin = 0;
ymin = 0;
xmax = screenWidth;
ymax = screenHeight;
```

World Coordinates

- More typically, we place the origin in world coordinates at the center of the screen.
- Then we assign new boundaries accordingly.
- For example,

```
xmin = -4.0;
xmax = 4.0;
ymin = -3.0; // Aspect ratio 4/3
ymax = 3.0;
```

Resizing the Window

- When we resize the window, the two most common ways to redraw the scene are
  - Distort the scene proportionally.
  - Keep the scene a constant size and keep the upper-left corner fixed.

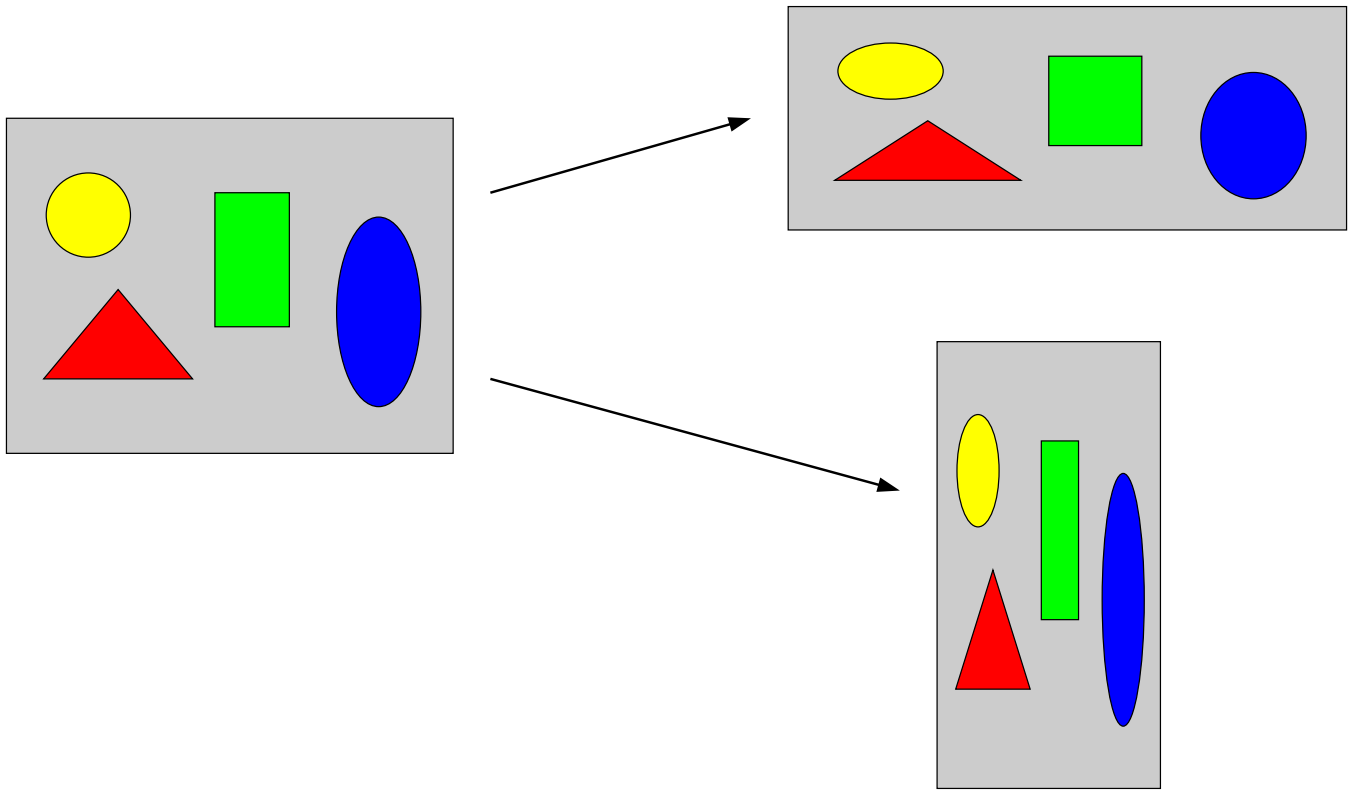
Outline

Distorting the Scene

- If we change the viewport without changing the world-coordinates boundaries, then the scene will be distorted.
- That is, if we do not recompute `xmin`, `xmax`, `ymin`, and `ymax`, the scene will be distorted.

Distorting the Scene

- The scene will be stretched or contracted in both the x and y directions.



Distorting the Scene

Example (Distorting the Scene)

- The code.
- The executable.

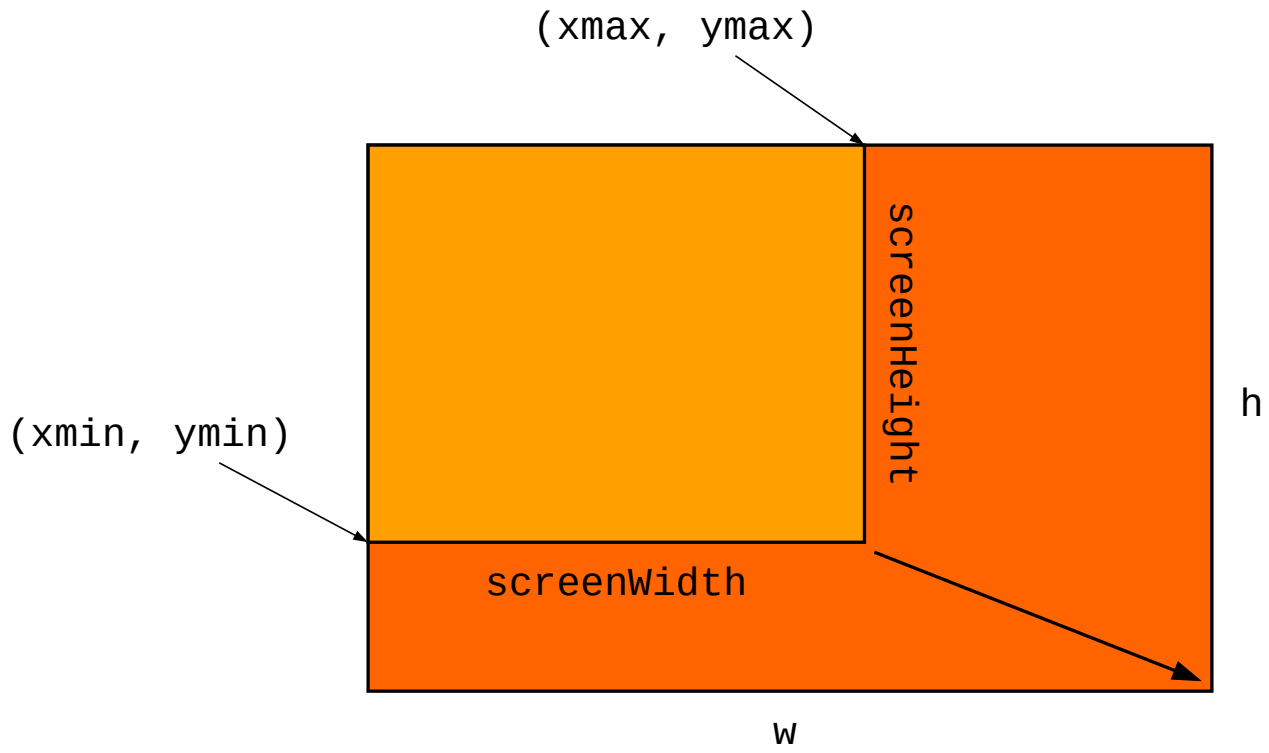
Outline

Constant Size, Fixed Corner

- We may want to keep the objects in the window the same size whether the window size is increased or decreased.
- We simply show more or less of the scene.
- The question is, what part of the scene should be added or deleted?

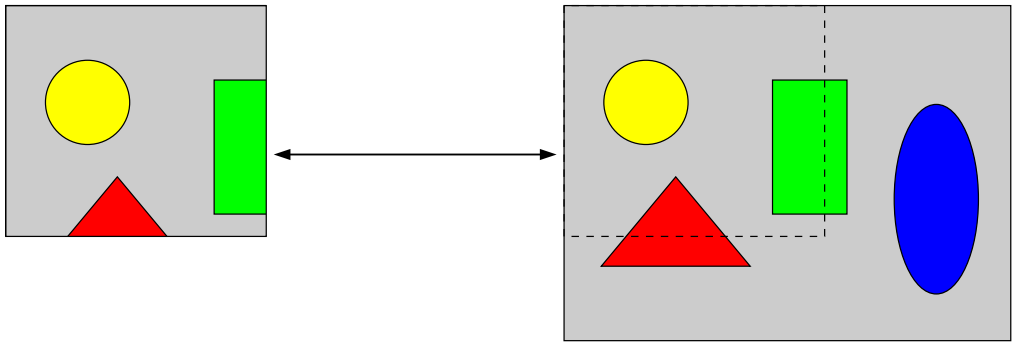
Constant Size, Fixed Corner

- For simplicity, assume that the window border is pulled down and to the right.



Constant Size, Fixed Corner

- Many applications keep the upper-left corner of the scene fixed.
- Thus, `xmin` and `ymax` remain unchanged.
- `xmax` and `ymin` must be recomputed.



Constant Size, Fixed Corner

Example (Constant Size, Fixed Corner)

- The code.
- The executable.

Constant Size, Fixed Corner

- In the *x*-direction, the extent has been changed by the fraction `w/screenWidth`.
- Therefore,

```
xmax = xmin + (w/screenWidth)
                *(xmax - xmin);
```

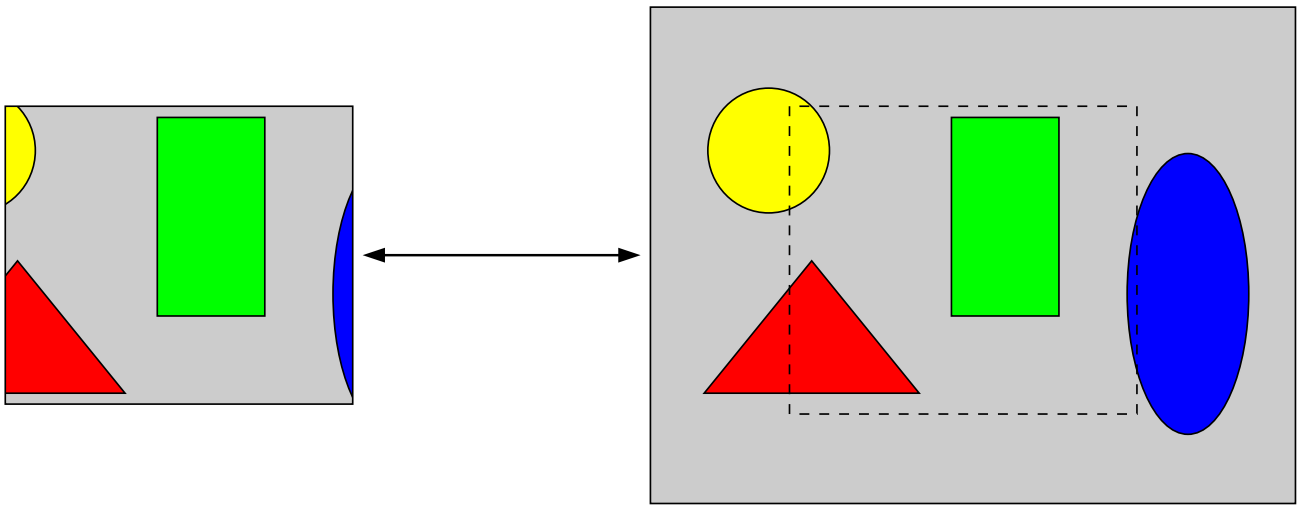
- Similarly,

```
ymin = ymax - (h/screenHeight)
                *(ymax - ymin);
```

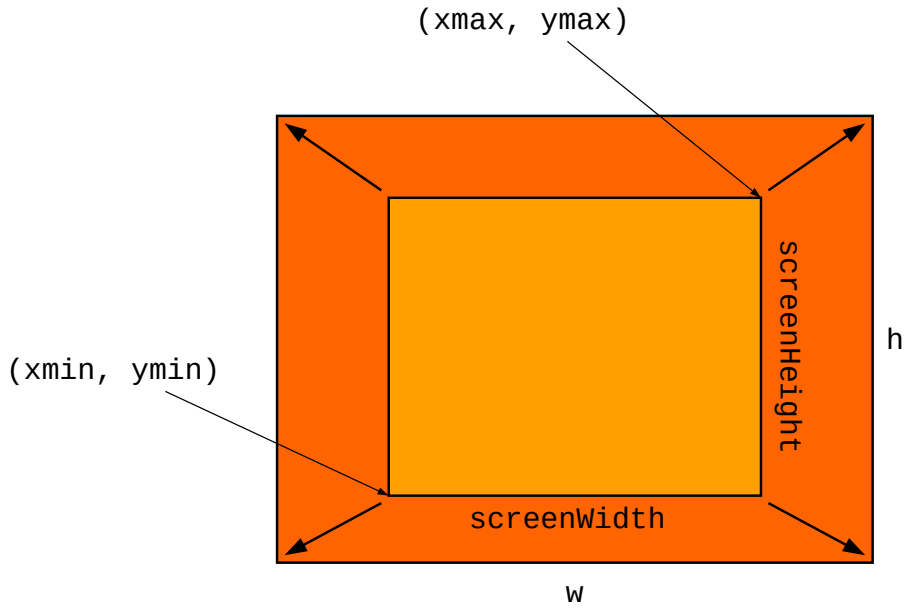
Outline

Constant Size, Fixed Center

- Sometimes we may want to expand the scene equally in all directions, keeping the center point in the center.



Constant Size, Fixed Center



Constant Size, Fixed Center

Example (Constant Size, Fixed Center)

- The code.
- The executable.

Constant Size, Fixed Center

- The old center was at

```
((xmin + xmax)/2, (ymin + ymax)/2)
```

- The new screen width is

```
w/screenWidth*(xmax - xmin)
```

Constant Size, Fixed Center

- Half of this amount should be added to and subtracted from the center to give the new `xmax` and `xmin`.

```
xmin = (xmax + xmin)/2
       - (w/screenWidth)*(xmax - xmin)/2;
xmax = (xmax + xmin)/2
       + (w/screenWidth)*(xmax - xmin)/2;
```

- Similar calculations give `ymin` and `ymax`.

Homework

Homework

- Read Section 2.7 – the `glViewport()` function.