Drawing and Coordinate Systems

Coordinate Systems
- Screen Coordinate system
- World Coordinate system
- World window
- Viewport
- Window to viewport mapping

Screen Coordinate System
- 2D Regular Cartesian Grid
- Origin (0,0) at lower left corner (OpenGL convention)
- Horizontal axis – x
  - Vertical axis – y
- Pixels are defined at the grid intersections
- This coordinate system is defined relative to the display window origin
  (OpenGL: the lower left corner of the window)
World Coordinate System

- Screen coordinate system is not easy to use

Another example:
plot a sinc function:
$$sinc(x) = \frac{\sin(\pi x)}{\pi x}$$
$$x = -4 \ldots +4$$

It would be nice if we can use application specific coordinates – world coordinate system

```c
glBegin(GL_LINE_STRIP);
  for (x = -4.0; x < 4.0; x += 0.1) {
    GLfloat y = sin(3.14 * x) / (3.14 * x);
    glVertex2f (x, y);
  }
glEnd();
```

Define a world window
World Window

- World window – a rectangular region in the world that limits our view

Define by

\[ W_L, W_R, W_B, W_T \]

Use OpenGL command:

\[
\text{gluOrtho2D(left, right, bottom, top)}
\]

Viewport

- The rectangular region in the screen that maps to our world window

Defined in the window’s (or control’s) coordinate system

\[
\text{glViewport(int left, int bottom, int (right-left), int (top-bottom));}
\]

To draw in world coordinate system

- Two tasks need to be done
  - Define a rectangular world window (call an OpenGL function)
  - Define a viewport (call an OpenGL function)
  - Perform window to viewport mapping (OpenGL internals will do this for you)

A simple example

```c
DrawQuad()
{
    glViewport(0,0,300,200);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho2D(-1,1,-1,1);
    glBegin(GL_QUADS);
    glColor3f(1,1,0);
    glVertex2i(-0.5,-0.5); glVertex2i(+0.5,-0.5); glVertex2i(+0.5,+0.5); glVertex2i(-0.5,+0.5);
    glEnd();
}
```
The objects in the world window will then be drawn onto the viewport. How to calculate \((sx, sy)\) from \((x, y)\)?

First thing to remember – you don’t need to do it by yourself. OpenGL will do it for you.
- You just need to define the viewport (with `glViewport()`), and the world window (with `gluOrtho2D()`)
- But we will look ‘under the hood’

A practical OpenGL issue
- Before calling `gluOrtho2D()`, you need to have the following two lines of code –

```c
glMatrixMode(GL_PROJECTION);
gluOrtho2D(Left, Right, Bottom, Top);
```
Window to viewport mapping

- Things that are given:
  - The world window \( (W_L, W_R, W_B, W_T) \)
  - The viewport \( (V_L, V_R, V_B, V_T) \)
  - A point \((x, y)\) in the world coordinate system
- Calculate the corresponding point \((sx, sy)\) in the screen coordinate system

Basic principle: the mapping should be proportional

\[
\begin{align*}
\frac{x - W_L}{W_R - W_L} &= \frac{sx - V_L}{V_R - V_L} \\
\frac{y - W_B}{W_T - W_B} &= \frac{sy - V_B}{V_T - V_B}
\end{align*}
\]

\[
\begin{align*}
sx &= x \cdot \frac{V_R - V_L}{W_R - W_L} - W_L \cdot \frac{V_R - V_L}{W_R - W_L} + V_L \\
sy &= y \cdot \frac{V_T - V_B}{W_T - W_B} - W_B \cdot \frac{V_T - V_B}{W_T - W_B} + V_B
\end{align*}
\]

Some practical issues

- How to set up an appropriate world window automatically?
- How to zoom in the picture?
- How to set up an appropriate viewport, so that the picture is not going to be distorted?
World window setup

- The basic idea is to see all the objects in the world
  - This can just be your initial view, and the user can change it later
- How to achieve it?

World window set up

- Find the world coordinates extent that will cover the entire scene (the bounding box)

 Zoom into the picture

Shrink your world window – call gluOrtho2D() with a new range

Non-distorted viewport setup

- Distortion happens when …
- World window and display window have different aspect ratios
- Aspect ratio?
- $R = W / H$
Fixing the aspect ratio

- Method I – Fixed camera view
  - Limit the viewport to a portion of the window. (covered next)
  - Constrain the user’s resizing ability.
  - Adjust the window (or control) size.
- Method II – Adjusting the scale to compensate for a non-square window.
  - We will cover this when we look at 3D.

Match aspect ratios

- World window
  - Aspect Ratio = R
- Display window
  - Aspect Ratio = W / H

\[ R > \frac{W}{H} \]
Compare aspect ratios

World window
Aspect Ratio = R

Display window
Aspect Ratio = W / H

R < W / H

Match aspect ratios

World window
Aspect Ratio = R

Display window
Aspect Ratio = W / H

R < W / H

Match aspect ratios

World window
Aspect Ratio = R

Display window
Aspect Ratio = W / H

R < W / H

When to call glViewport()?

- Initialization
- When the user resizes the display window.
- New type of camera? 35mm, 70mm, ...

Note: Resize event is actually called on initialization, but your callback may not have been connected at this time.
Resize event

Void resize(int W, int H)
{
    glViewport(0,0,W, H);
}

You can provide your own to make sure the aspect ratio is fixed.

Put it all together

DrawQuad()
{
    glViewport(0,0,300,200);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho2D(-1,1,-1,1);
    glBegin(GL_QUADS);
        glColor3f(1,1,0); glVertex2i(-0.5,-0.5); glVertex2i(+0.5,-0.5); glVertex2i(+0.5,+0.5); glVertex2i(-0.5,+0.5);
    glEnd();
}

More Viewports

- Viewports can also be thought of as clip windows. This can be useful for:
  - User interaction
  - Static camera – small moving object
  - Limited field-of-view
  - Occlusion culling
  - Selection (picking)