Windows and Viewports

Previously we looked at an OpenGL window where x and y were plotted as positive pixel values.
However, we may not be interested in keeping track of pixels like this.
We may prefer to plot points in the coordinates in which they are given.
This means handling both positive and negative values.

World windows and viewports

So far we work with:
- positive values only;
- the values must extend over a large range.

- We may not want to think in terms of pixels, but in terms of varying x or y from a negative value to a positive value.
- World coordinates, world window
- Viewport, automatic change of coordinates.

\[
sin(x) = \frac{\sin(\pi x)}{\pi}
\]
Windows and Viewports

- The world window is a rectangle.
- The viewport is a rectangle.
- Both are not necessarily the same size or have the same aspect ratio.
- Coordinates need to be stretched, shrunk and moved to make them fit.

Definitions

Viewport:
- A rectangular viewport is defined in the screen window.
- Mapping between the world window and the viewport.
- The window and viewport are both aligned rectangles specified by the programmer.

Proper Scaling & Shifting

- Establishing a suitable mapping between them.
- World window and viewport are both aligned rectangles specified by the programmer.

This is called Mapping
Windows and Viewports

• Mapping involves scaling and translation (moving).
• Both the world window and viewport can be any aligned rectangle.
• Usually the viewport is set to take up the entire screen window.

Dot Plots: scaling and shifting

Scaling x: \[ x' = x \times \text{screenWidth} / 4.0 \]
Scaling y: \[ y' = (y + 1.0) \times \text{screenHeight} / 2.0 \]

Windows and Viewports

x' = Ax + B
y' = Cy + D

This is exactly how mapping is achieved!!
Windows and Viewports

- Just like the plotting of the function from the last lecture, mappings are created using proportions.
- The distance a point is along the x axis in the World Window will be proportional to the distance the point is plotted along the x axis in the Viewport.
- How far is x from W.l?
  \[ \text{dx} = x - W.l \]
- As a proportion of the whole length it is
  \[ \frac{\text{dx}}{\text{dsx}} = \frac{x - W.l}{W.r - W.l} \]

Window-to-viewport mapping

\[
\begin{align*}
\frac{x - W.l}{W.r - W.l} &= \frac{y - V.t}{V.r - V.t} \\
\frac{\text{dx}}{\text{dsx}} &= \frac{\text{dy}}{\text{dsy}}
\end{align*}
\]

Windows and Viewports

\[
\begin{align*}
x_{min}, y_{min}, x_{max}, y_{max} &= W.l, W.b, W.r, W.t \\
\text{SCREENWIDTH} &= V.r - V.l \\
\text{SCREENHEIGHT} &= V.t - V.b
\end{align*}
\]

Window-to-viewport transformations

\[
\begin{align*}
\text{sx} &= A \cdot x + C, \quad \text{sy} = B \cdot y + D \\
\text{with}
\begin{align*}
A &= \frac{V.r - V.l}{W.r - W.l}, \quad C = V.l - A \cdot W.l \\
B &= \frac{V.t - V.b}{W.t - W.b}, \quad D = V.b - B \cdot W.b
\end{align*}
\]

Window To Viewport Transformations

For some constants A, B, C, and D:
- A and B scale the x and y coordinates
- C and D shift them

\[
\begin{align*}
S_x &= A \cdot C \\
S_y &= B \cdot D
\end{align*}
\]
Example 3.2.1

Figure 3.6: An example of a window and viewport. The window has [W, W, W, W]. The viewport has [V, V, V, V].

If this is correct, (-10, -6) should map to (0, 0) and (10, 6) -> (600, 400).

sx = A * x + C
sy = B * y + D

Example 3.2.1

Windows and Viewports

Example

A = 600 / 20 = 30
B = 400 / 20 = 33.3333
C = 0 - 30 * -10 = 300
D = 0 - 33.3333 * -6 = 200

If this is correct, (-10, -6) should map to (0, 0) and (20, 6) -> (600, 400).

sx = A * x + C
sy = B * y + D

Windows and Viewports

• Do you need to perform these calculations each time you draw something with OpenGL??

Windows and Viewports

• No

• OpenGL does all the hard work for you.

• But it is important that you understand what is going on....

Windows and Viewports

• Each time you call for a vertex to be drawn (e.g. glVertex2f() etc.) the coordinates of the point are passed through a set of transformations that map world coordinates into viewport coordinates.

```c
#include <GL/glut.h>

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(600, 400);
    glutCreateWindow("Example 3.2.1");
    glutDisplayFunc(drawWindow);
    glutMainLoop();
    return 0;
}

void drawWindow()
{
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glOrtho(0.0, 2.0, 0.0, 1.0, 0.0, 1.0);
    glBegin(GL_TRIANGLES);
    glVertex2f(-1.0, -0.5);
    glVertex2f(1.0, -0.5);
    glVertex2f(0.0, 0.5);
    glEnd();
}
```
Windows and Viewports

- First set the world window coordinates with:
  ```
  void glOrtho2D(GLDouble left, GLDouble right, GLDouble bottom, GLDouble top);
  ```
- Then set the viewport with:
  ```
  void glViewport(GLint xmin, GLint ymin, GLint width, GLint height);
  ```

```c
//--------------- setWindow ---------------------
void setWindow(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top) {
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(left, right, bottom, top);
}
```

### Handy Functions

```c
//------------ setViewport ---------------------
void setViewport(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top) {
    glViewport(left, bottom, right - left, top - bottom);
}
```
Example

\[
\sin(x) = \frac{\sin(\pi x)}{\pi x}
\]

```c
void myDisplay(void)
{
    glBegin(GL_LINE_STRIP);
    for(GLfloat x = -4.0; x < 4.0; x += 0.1)
    {
        GLfloat y = sin(3.14159 * x) / (3.14159 * x);
        glVertex2f(x, y);
    }
    glEnd();
}
```

Windows and Viewports

- Plotting a function revisited.

```
// set the viewing coordinates
setWindow(xmin, xmax, ymin, ymax);
setViewport(0, 640, 0, 480);
```

```
// plotting a function
for( GLdouble x = xmin; x < xmax; x += 0.005)
{
    glVertex2d(x, pow(2.7183, -x)*cos(2*3.14*x));
}
```

What we used before?

- Orthographic view

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)
```
Set the Viewport

- Recall, if the aspect ratio of a rectangle is less than 0, the rectangle is taller than wide.
  - E.g. \( W/H = 3/5 \)
- If the aspect ratio is greater than 0, the rectangle is wider than tall.
  - E.g. \( W/H = 5/3 \)

Set the Viewport

a) \( R > W/H \) (where \( R \) is the aspect ratio of the world window)
- If the world window is 'flatter' than the screen window, there will be unused space above and/or below.
- The width of the world window will be mapped to the entire width of the screen window.
  
  \( setViewport(0, W, 0, W/R); \)

b) \( R < W/H \) (where \( R \) is the aspect ratio of the world window)
- If the world window is 'taller' than the screen window, there will be unused space on the sides.
- The height of the world window will be mapped to the entire height of the screen window.
  
  \( setViewport(0, H*R, 0, H); \)

Set the Viewport

- If the world window is 'flatter' than the screen window, there will be unused space above and/or below.
- The width of the world window will be mapped to the entire width of the screen window.
  
  \( setViewport(0, W, 0, W/R); \)

- If the world window is 'taller' than the screen window, there will be unused space on the sides.
- The height of the world window will be mapped to the entire height of the screen window.
  
  \( setViewport(0, H*R, 0, H); \)

Resizing the screen window, and the resize event

In a windows-based system the user can resize the screen window at run-time, typically by dragging one of its corners with the mouse. This action generates a resize event that the system can respond to.

Matching the viewport

```c
void myReshape(GLsizei W, GLsizei H) {
  glutReshape(0, W, 0, H);
  glutReshape(GLUT_RESIZE, 0, W, 0, H);
}
```

When it is executed the system automatically passes it the new width and height of the screen window, which it can use in its calculations.

Tiling

If we lay a number of copies of a figure side by side to cover the entire screen window it's called tiling the screen window. The picture that is copied at different positions is often called a motif. Tiling a screen window is easily achieved by using a different viewport for each instance of the motif.

```c
for(int i = 0; i < 5; i++)
  for(int j = 0; j < 5; j++)
    glViewport(i * 64, j * 64, 64, 64);
```