3. OpenGL - II

3.1 World Coordinate System, World Window, & Viewport

- Using the **device coordinate system** (DCS) directly is not flexible for many applications. Why?
  - Can deal with integers only
  - There is a maximum on the range of $x$ and $y$

- **Device-independent approach:**
  - Do the drawing in a **World Coordinate System** (WCS)
  - Use **world window** to define the region to be shown
  - Use **viewport** (a rectangular region of the screen window) to show the drawing
Illustration:

- Need a **window-to-viewport** mapping
- The mapping preserves **aspect ratio** \( \frac{\text{width}}{\text{height}} \)
- **clipping**: anything that is outside the world window should be discarded before the mapping
- **Clipping** and **mapping** are performed by OpenGL
- Example: plot \( \text{sinc} \left( x \right) = \frac{\sin(\pi x)}{\pi x} \) between \( x = -4 \) and \( x = 4 \) in the viewport \((0, 640, 0, 480)\).
Ideal condition: write the code the following way and let the system worry about the mapping (transformation)

```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 ( );

    glFlush ( );
}
```

How?
Window-to-Viewport Mapping:
- Preserving proportions

\[
\frac{sx - V.l}{V.r - V.l} = \frac{x - W.l}{W.r - W.l}
\]

and

\[
\frac{sy - V.b}{V.t - V.b} = \frac{y - W.b}{W.t - W.b}
\]
Hence,

\[
\begin{align*}
    sx &= A \cdot x + C \\
    sy &= B \cdot y + D
\end{align*}
\]

where

\[
A = \frac{V.r - V.l}{W.r - W.l}, \quad C = V.l - A \cdot W.l
\]

\[
B = \frac{V.t - V.b}{W.t - W.b}, \quad D = V.b - B \cdot W.b
\]
Doing it in OpenGL:

Set Window:

\[
glMatrixMode ( \text{GL_PROJECTION} ) ; \\
glLoadIdentity ( ) ; \\
gluOrtho2D ( W\_left, W\_right, W\_bottom, W\_top ) ;
\]

Set Viewport:

\[
glViewport ( V\_left, V\_bottom, V\_width, V\_height )
\]
Example:

```c
void myDisplay ( void )
{
    glClear ( GL_COLOR_BUFFER_BIT ); // clear the screen
    //
    glMatrixMode ( GL_PROJECTION );
    glLoadIdentity ( );
    gluOrtho2D ( -5.0, 5.0, -3.0, 1.0 ); // set the window
    //
    glViewport(0, 0, 640, 480); // set the viewport
    //
    glBegin ( GL_LINE_STRIP );
    for ( GLfloat x = -4.0; x < 4.0; x += 0.1 )
    { // draw the plot
        glVertex2f(x, sin(3.14159 * x) / (3.14159 * x));
    }
    glEnd ( );
    glFlush ( );
}
```
3.2 A Few Applications:

1. Tile the screen window
   - Use a different viewport for each instance of the pattern

```c
//*******************************************
void myDisplay(void)
{
    glClear ( GL_COLOR_BUFFER_BIT );
    glMatrixMode ( GL_PROJECTION );
    glLoadIdentity ( ) ;
    gluOrtho2D ( 0.0, 640.0, 0.0, 480.0 );

    for (int i=0; i < 10; i++)
        for (int j=0; j < 11; j++) {
            glViewport ( i*64, j*44, 64, 44);

            // Redraw the plot
            glBegin ( GL_LINE_STRIP );
            for ( GLfloat x = -4.0; x < 4.0; x += 0.1 )
                glVertex2f ( x, sin(3.14159 * x) / (3.14159 * x ) );
            glEnd ( );
        }
    glFlush();
}
```
2. Flip an image upside down
   - Simply flip the window upside down

```c
//**************************************************
void myDisplay ( void )
{
  glClear ( GL_COLOR_BUFFER_BIT );
  //
  setWindow ( -5.0, 5.0, -0.3, 1.0 );
  //
  for ( int i=0; i < 10; i++ )
    for ( int j=0; j < 11; j++ ) {
      if ( ( i+j)%2 == 0 )
        setWindow ( -5.0, 5.0, -0.3, 1.0 );
      else
        setWindow ( -5.0, 5.0, 1.0, -0.3 );
      glViewport ( i*64, j*44, 64, 44 );
      glBegin ( GL_LINE_STRIP );
      for ( GLfloat x = -4.0; x < 4.0; x += 0.1)
        glVertex2f ( x, sin(3.14159 * x) / (3.14159 * x));
      glEnd ( );
    }
  glFlush ( );
}
```
3. Zooming effect
   - hold the viewport but reduce (zoom in) or increase (zoom out) the dimension of the window

//********************************************************************************
void myDisplay(void)
{
    float cx = 0.0, cy = 0.3; // center of the window
    float H, W = 5.0, aspect = 7.143;
    int NumFrames = 200;

    glClear(GL_COLOR_BUFFER_BIT);       // clear the screen
    setViewport(0, 640, 0, 480);        // set the viewport
    for(int frame = 0; frame < NumFrames; frame++)
    {
        glClear(GL_COLOR_BUFFER_BIT);    // clear the screen
        W *= 0.995;                      // reduce the window width
        H = W / aspect;                 // maintain the same aspect ratio
        setWindow(cx - W, cx + W, cy - H, cy + H);
        //set the next window
        drawSincFunc ( );
        // glutSwapBuffers ( );
    }
Problems with this approach

- You get flickering, because some portions of the image can be viewed for only very short period of time.

How to achieve smooth animation?

- Use **double buffering**

**How?**

1. Use "GLUT_DOUBLE" instead of "GLUT_SINGLE" in

   ```c
   glutInitDisplayMode ( xxxx | GLUT_RGB );
   ```

2. Include the following instruction at the end of "myDisplay ()".

   ```c
   glutSwapBuffers ( );
   ```
**Line clipping:** (Cohen-Sutherland algorithm)

- To avoid unnecessary computation, perform tests on trivially accepted cases and trivially rejected cases first.
- If both endpoints are inside the window, then the line segment is inside the window.
- If both endpoints are to the left ($x < x_{\text{min}}$), to the right ($x > x_{\text{min}}$), below ($y < y_{\text{min}}$), or above ($y > y_{\text{min}}$) the window, then the line segment is outside the window.
To perform the tests efficiently, divide the world coordinate system into 9 regions and assign each of them a four-bit code.

<table>
<thead>
<tr>
<th>1001</th>
<th>1000</th>
<th>1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>bottom</td>
<td>right</td>
<td>left</td>
</tr>
</tbody>
</table>

bit 1: sign bit of \((x - x_{\text{min}})\)
bit 2: sign bit of \((x_{\text{max}} - y)\)
bit 3: sign bit of \((y - y_{\text{min}})\)
bit 4: sign bit of \((y_{\text{max}} - y)\)
The Cohen-Sutherland Algorithm

1. Compute the codes for the endpoints of the line segment to be clipped

2. Repeat until the line segment is either trivially accepted or rejected

2.1 [Trivial Acceptance Test]
   If bitwise OR of the codes is 0000 (line segment is inside the window), draw the line segment and stop.

3. [Trivial Rejection Test]
   If bitwise AND of the codes is not 0000 (line segment is outside the window), discard the line segment and stop.

4. [Subdivide the segment]

   4.1 Pick an endpoint whose code is non-zero (the endpoint that is outside the window)

   4.2 Find the first non-zero bit: this corresponds to the window edge which intersects the line segment

   4.3 Compute the intersection point and replace the outside endpoint with the intersection point
An Example

Use bit 2 of $A$ (right clipping edge) to do the subdivision
Subdivide at $C$ (Find $y$ coordinate of $C$)
$y = m \cdot x_{\text{max}} + b$
Use bit 4 of $C$ (top clipping edge) to do the subdivision

Subdivide at $D$ (need to find $x$ coordinate of $D$)

$$x = (y_{\text{max}} - b)/m$$
Use bit 1 of B (left clipping edge) to do the subdivision

Subdivide at $E$ (need to find $y$ coordinate of $E$)

$$y = m \cdot x_{\text{min}} + b$$
Example (con’t)

Segment $ED$ is trivially accepted