Chapter 8 (I)
Two - Dimensional Viewing
Outline

- Viewing Pipeline
- World Coordinates Transfer to Viewing Coordinates
- Normalization and Viewport Transformations
- OpenGL 2D Viewing Functions
- OpenGL 2D Viewing Program Example
Viewing Pipeline

1. Modelling Transform
2. Viewing Transform
3. Clip to a View Volume
4. Projection Transform
5. Viewport Mapping
Viewing Volume

- Viewing volume
  - A closed volume which delimits the infinite 3D space to finite volume.
  - Points outside it will not appear on the screen.

- Two projections to create viewing volume
  - Orthographic projection
    - Objects rendered are not affected by the distance
    - E.g.: a menu, a text on a screen, 2D objects…
  - Perspective projection
    - Objects rendered are affected by the distance
    - E.g.: a car is seen smaller when it move away

(From OpenGL Super Bible)
Viewing Volume

- Orthographic projection
  - Viewing volume shape is a parallelepiped (平行六面體).
  - Parallel clipping planes

- Perspective projection
  - Viewing volume shape is a truncated pyramid (its top is cut).
  - Non-parallel side clipping planes
Viewport

- Viewport
  - 2D drawing region of the screen where the final result is mapped.
Viewport

- Measured in actual window coordinates

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

- \((x, y)\): specifies the lower-left corner of the viewport;
- \(width\) and \(height\): the size of the viewport rectangle.

**By default**, the initial viewport are \((0, 0, \text{winWidth}, \text{winHeight})\), where \(\text{winWidth}\) and \(\text{winHeight}\) are the size of the window.

(From OpenGL Super Bible)
Viewport

- Example

*From OpenGL Super Bible*
To make the viewing process independent of any output device, viewing coordinates is converted to normalized coordinates.

Clipping is usually performed in normalized coordinates.
• Matrix-form in OpenGL
  • The viewing and modeling transformations you specify are combined to form the modelview matrix, which is applied to the incoming object coordinates to yield eye (viewing) coordinates.
  • The projection matrix to yield clip coordinates.
  • Defines a viewing volume
Matrix form in OpenGL (cont.)

- The **perspective division** is performed by dividing coordinate values by \( w \), to produce **normalized device coordinates**.
- The transformed coordinates are converted to **window coordinates** by applying the viewport transformation.
  - You can specify the size of the viewport to cause the final image result.
Stages of Vertex Transformation

- The viewing, modeling, and projection transformations matrix in OpenGL: 4×4 matrix \( M \) [in 2D, \( z = 0 \)]
- They are multiplied by the coordinates of each vertex \( v \) in the scene
  \[ v' = Mv \]
World Coordinates Convert to Viewing Coordinates

We can set up a 2D viewing coordinate system in the world coordinate frame.

\[(x_0, y_0)\]: an origin

yview: 2D view up vector

This viewing coordinate frame provides a reference for specifying the clipping window.

Fig. 8-3 A rotated world window in Viewing Coordinates.
World Coordinates Convert to Viewing Coordinates

![Diagram showing the conversion of World to Viewing Coordinates]

\[
M_{\text{we} \rightarrow \text{ve}} = R \cdot T \tag{8-1}
\]

Where \( T \) is the translation matrix that takes the viewing origin point \( P_0 \) to the world origin, and \( R \) is the rotation matrix that aligns the axes of the two reference frames.

**FIGURE 8-4** A Viewing-Coordinate frame is moved into coincidence with the World-Coordinate frame by
(a) applying a **translation** matrix \( T \) to move the viewing origin to the world origin, then (b) applying a **rotation** matrix \( R \) to align the axes of the two systems.
Normalization and Viewport Transformations

- Mapping a clipping window into a normalized viewport (the viewport is given within $([0,1], [0,1])$)

**FIGURE 8-6** A point $(x_w, y_w)$ in a world-coordinate clipping window is mapped to viewport coordinates $(x_v, y_v)$ within a unit square, so that the relative positions of the two points in their respective rectangles are the same.

$$\begin{align*}
\frac{x_v - x_{v,\text{min}}}{x_{v,\text{max}} - x_{v,\text{min}}} &= \frac{x_w - x_{w,\text{min}}}{x_{w,\text{max}} - x_{w,\text{min}}} \\
\frac{y_v - y_{v,\text{min}}}{y_{v,\text{max}} - y_{v,\text{min}}} &= \frac{y_w - y_{w,\text{min}}}{y_{w,\text{max}} - y_{w,\text{min}}}
\end{align*}$$
OpenGL 2D Viewing Functions

- OpenGL Projection Mode
  ```
  glMatrixMode (GL_PROJECTION);  //projection matrix
  glLoadIdentity ();
  ```

- GLU Clipping-Window Function
  ```
  gluOrtho2D (xwmin, xwmax, ywmin, ywmax);
  ```
  2D parallel projection.

- OpenGL Viewport Function
  ```
  glViewport (xvmin, yvmin, vpWidth, vpHeight);
  ```
  ```
  glGetInteger (GL_VIEWPORT, vpArray);
  ```
  To obtain the parameters for the currently active viewport: xvmin, yvmin, vpWidth, vpHeight.
OpenGL Viewport Function

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

Example: `gluOrtho2D (-2.0, 2.0, -2.0, 2.0);

You can change it by `glViewport()`.

Always rectangle;

x, y are in window coordinates

- Example: `gluOrtho2D (-2.0, 2.0, -2.0, 2.0);

`glViewport (0, 0, 300, 150);`

World Coordinates

Normalized Device Coordinates

Window Coordinates
OpenGL Viewport Function

• The rectangle area has an aspect ratio: width / height
  • Windows
    • glutInitWindowSize ( width, height );
  • 2D clipping window
    • gluOrtho2D ( left, right, bottom, top );
  • Viewport
    • glViewport ( x, y, width, height);

• In general, the clipping window (viewing volume) and viewport need to have the same ratio.
OpenGL 2D Viewing Program Example

- Two views of a triangle in the xy plane shown in a split screen, with its centroid at the world-coordinate origin

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

class wcPt2D {
public:
    GLfloat x, y;
};
```
void init (void)
{
  /* Set color of display window to white. */
  glClearColor (1.0, 1.0, 1.0, 0.0);

  /* Set parameters for world-coordinate clipping window. */
  glMatrixMode (GL_PROJECTION);
  gluOrtho2D (-100.0, 100.0, -100.0, 100.0);

  /* Set mode for construction geometric transformation matrix. */
  glMatrixMode (GL_MODELVIEW);
}
void triangle (wcPt2D *verts)
{
    GLint k;

    glBegin (GL_TRIANGLES);
    for (k =0; k < 3; k++)
        glVertex2f (verts[k].x, verts[k].y);
    glEnd ();
}
OpenGL 2D Viewing Program Example

```c
void displayFcn (void) {
    /* Define initial position for triangle. */
    wcPt2D verts [3] = {{-50.0, -25.0}, {50.0, -25.0}, {0.0, 50.0}};

    glClear (GL_COLOR_BUFFER_BIT);  // Clear display window.
    glColor3f (0.0, 0.0, 1.0);      // Set fill color to blue.
    glViewport (0, 0, 300, 300);      // Set left viewport.
    triangle (verts);                // Display red rotated triangle.

    /* Rotate triangle and display in right half of display window. */
    glColor3f (1.0, 0.0, 0.0);      // Set fill color to red.
    glViewport (300, 0, 300, 300);    // Set right viewport.
    glRotatef (90.0, 0.0, 0.0, 1.0);  // Rotate about z axis.
    triangle (verts);                // Display red rotated triangle.

    glFlush (); //Force to execute all OpenGL functions
}
```
OpenGL 2D Viewing Program Example

```c
void main (int argc, char **argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (50, 50);
    glutInitWindowSize (600, 300);
    glutCreateWindow ("Split-Screen Example");

    init();
    glutDisplayFunc (displayFcn);
    glutMainLoop ();
}
```
OpenGL 2D Viewing Functions

• You need to handle the changes in the window size
  • Registering a window reshape callback
    • void glutReshapeFunc ( void (*func) (int width, int height) );
  • Defining the reshape callback function: pass the new width and height of the window
    • called before the first call to the display function,
    • and called automatically when the window is reshaped.
  • e.g. MyReshape ();

```c
void MyReshape ( int width, int height )
{
    /* update viewport */
    glViewport (…);
    /* reset viewing volume */
    glMatrixMode ( GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D (…);
    /* set modelview matrix mode */
    ...
}
```
Summary

- 2D viewing pipeline
- OpenGL 2D viewing functions