CS380: Computer Graphics
2D Imaging and Transformation

Sung-Eui Yoon
(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/CG

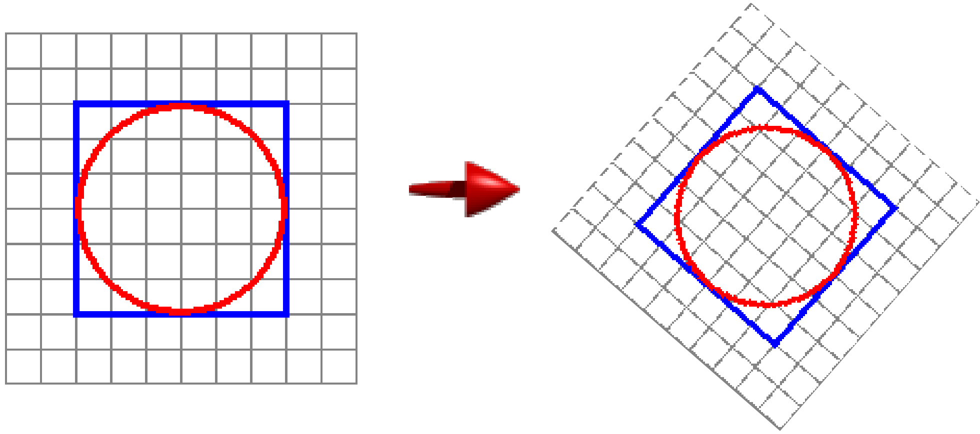
KAIST

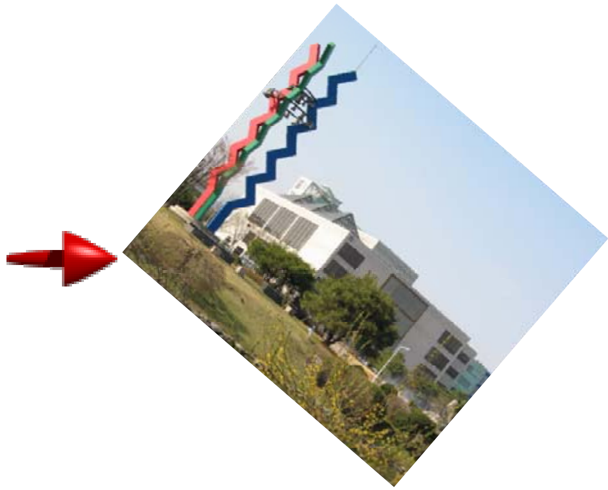
Class Objectives

- Write down simple 2D transformation matrixes
- Understand the homogeneous coordinates and its benefits
- Know OpenGL-transformation related API
- Implement idle-based animation method

2D Geometric Transforms

- Functions to map points from one place to another
- Geometric transforms can be applied to
  - Drawing primitives (points, lines, conics, triangles)
  - Pixel coordinates of an image





Demo

Translation

- **Translations have the following form:**

$$x' = x + t_x \qquad y' = y + t_y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- ***inverse function:*** **undoes the translation:**

$$x = x' - t_x \qquad y = y' - t_y$$

- ***identity*****: leaves every point unchanged**

$$x' = x + 0 \qquad y' = y + 0$$

2D Rotations

- **Another group - rotation about the origin:**

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = R \begin{bmatrix} x \\ y \end{bmatrix}$$

$$R^{-1} = \begin{bmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{bmatrix}$$

$$R_{\theta=0} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Rotations in Series

- **We want to rotate the object 30 degree and, then, 60 degree**

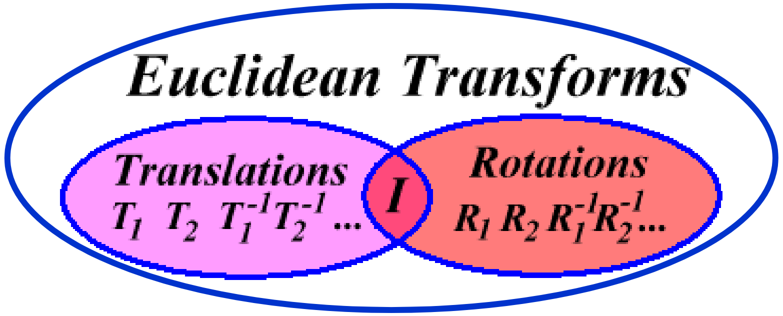
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(60) & -\sin(60) \\ \sin(60) & \cos(60) \end{bmatrix} \begin{bmatrix} \cos(30) & -\sin(30) \\ \sin(30) & \cos(30) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

We can merge multiple rotations into one rotation matrix

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(90) & -\sin(90) \\ \sin(90) & \cos(90) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Euclidean Transforms

- **Euclidean Group**
  - **Translations + rotations**
  - **Rigid body transforms**



- **Properties:**
  - **Preserve distances**
  - **Preserve angles**
  - **How do you represent these functions?**

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

KAIST

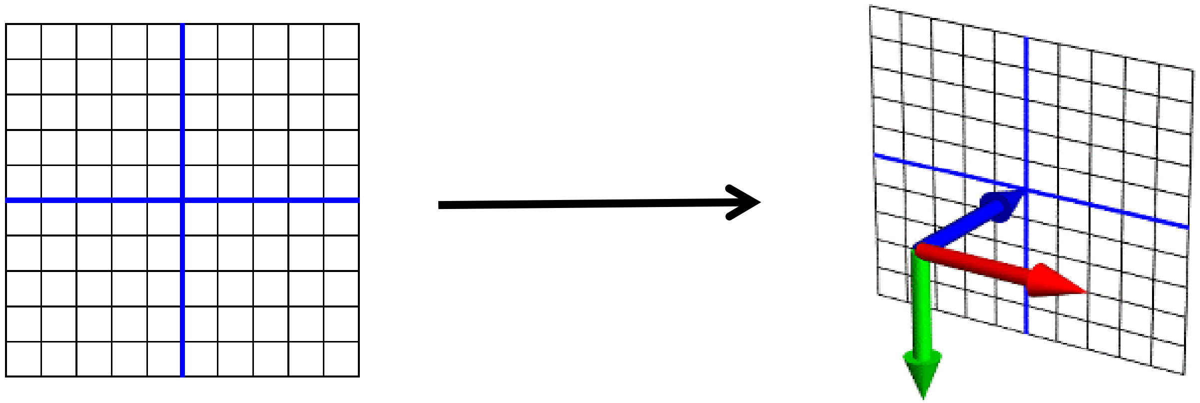
Problems with this Form

- Translation and rotation considered separately
  - Typically we perform a series of rotations and translations to place objects in world space
  - It's inconvenient and inefficient in the previous form
  - Inverse transform involves multiple steps

- How can we address it?
  - How can we represent the translation as a matrix multiplication?

Homogeneous Coordinates

- **Consider our 2D plane as a subspace within 3D**



(x, y) **(x, y, z)**

KAIST

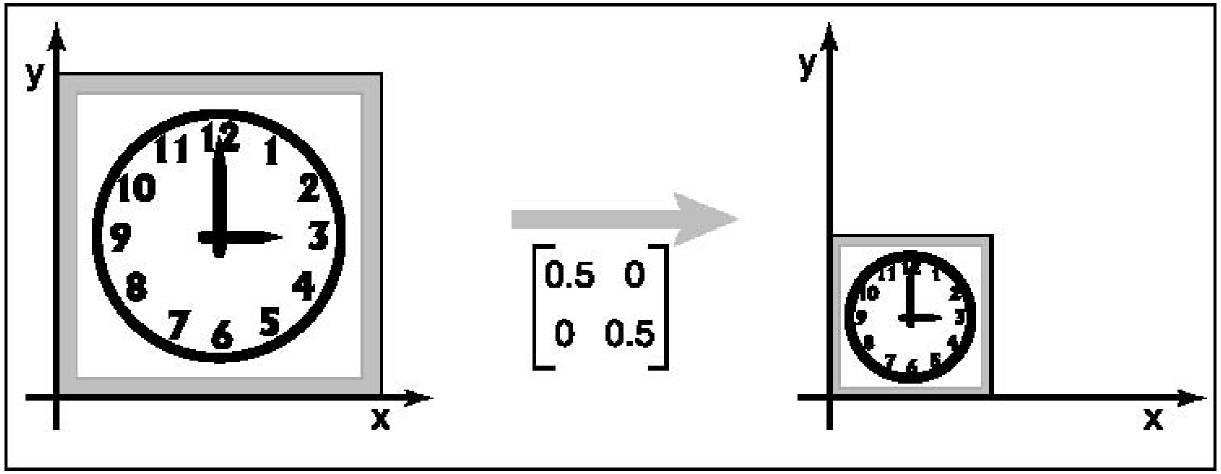
Matrix Multiplications and Homogeneous Coordinates

- **Can use any planar subspace that does not contain the origin**
- **Assume our 2D space lies on the 3D plane *z = 1***
  - **Now we can express all Euclidean transforms in matrix form:**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

KAIST

Scaling



- **S is a scaling factor**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

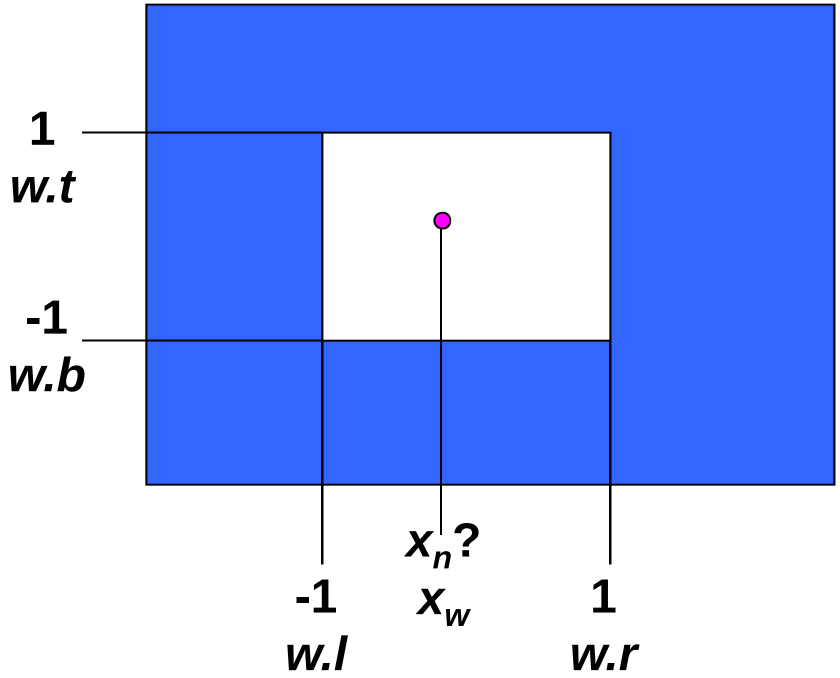
Example: World Space to NDC

$$\frac{x_n - (-1)}{1 - (-1)} = \frac{x_w - (w.l)}{w.r - w.l}$$

$$x_n = 2\frac{x_w - (w.l)}{w.r - w.l} - 1$$

$$x_n = Ax_w + B$$

$$A = \frac{2}{w.r - w.l}, \quad B = -\frac{w.r + w.l}{w.r - w.l}$$



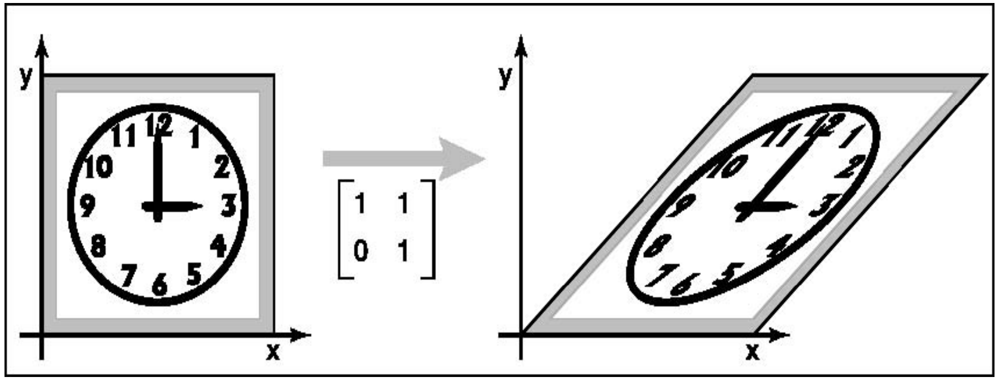
Example: World Space to NDC

- **Now, it can be accomplished via a matrix multiplication**
  - **Also, conceptually simple**

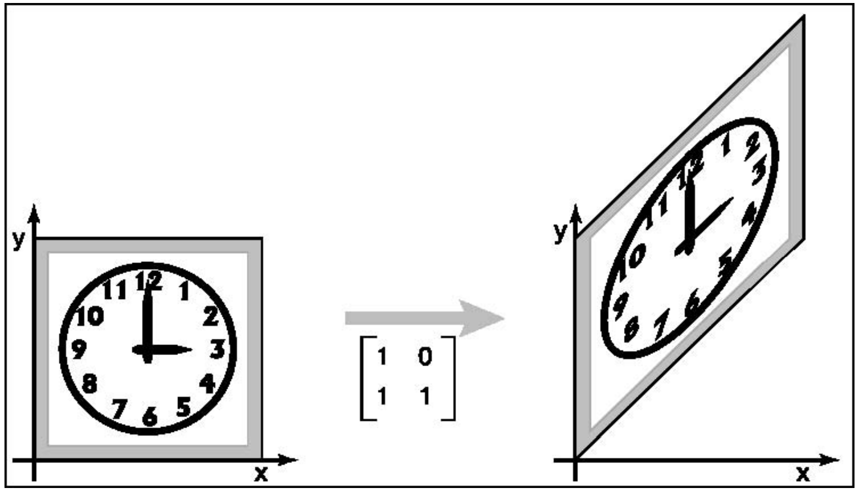
$$\begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{w.r - w.l} & 0 & -\frac{w.r + w.l}{w.r - w.l} \\ 0 & \frac{2}{w.t - w.b} & -\frac{w.t + w.b}{w.t - w.b} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix}$$

Shearing

- Push things sideways
- Shear along x-axis

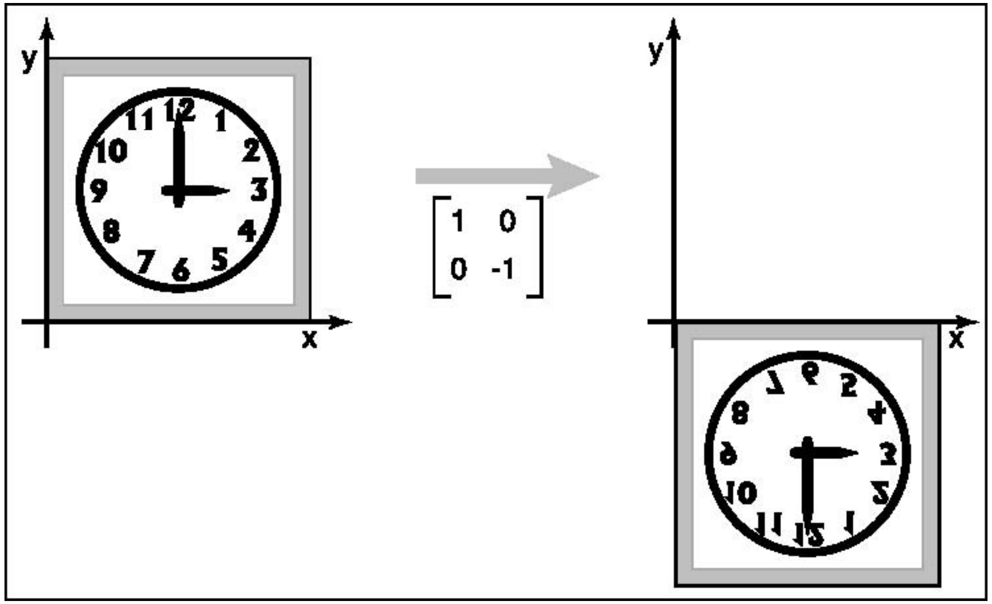


- Shear along y-axis

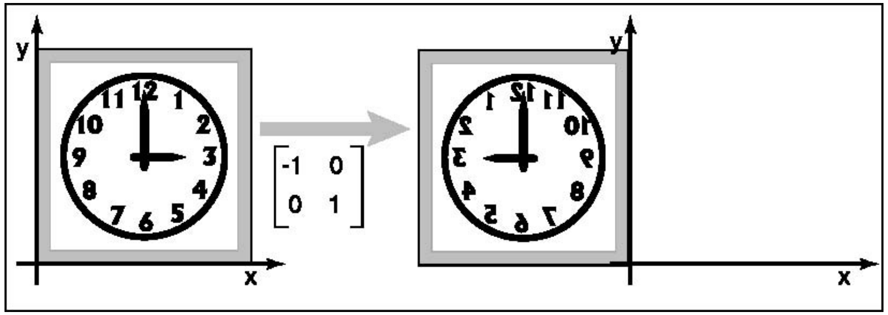


Reflection

- **Reflection about x-axis**

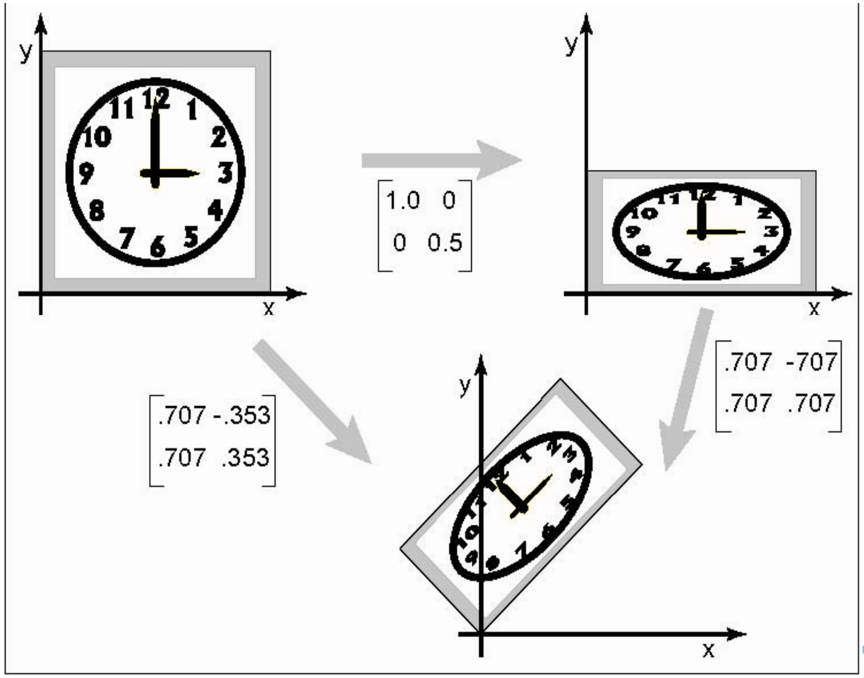


- **Reflection about y-axis**



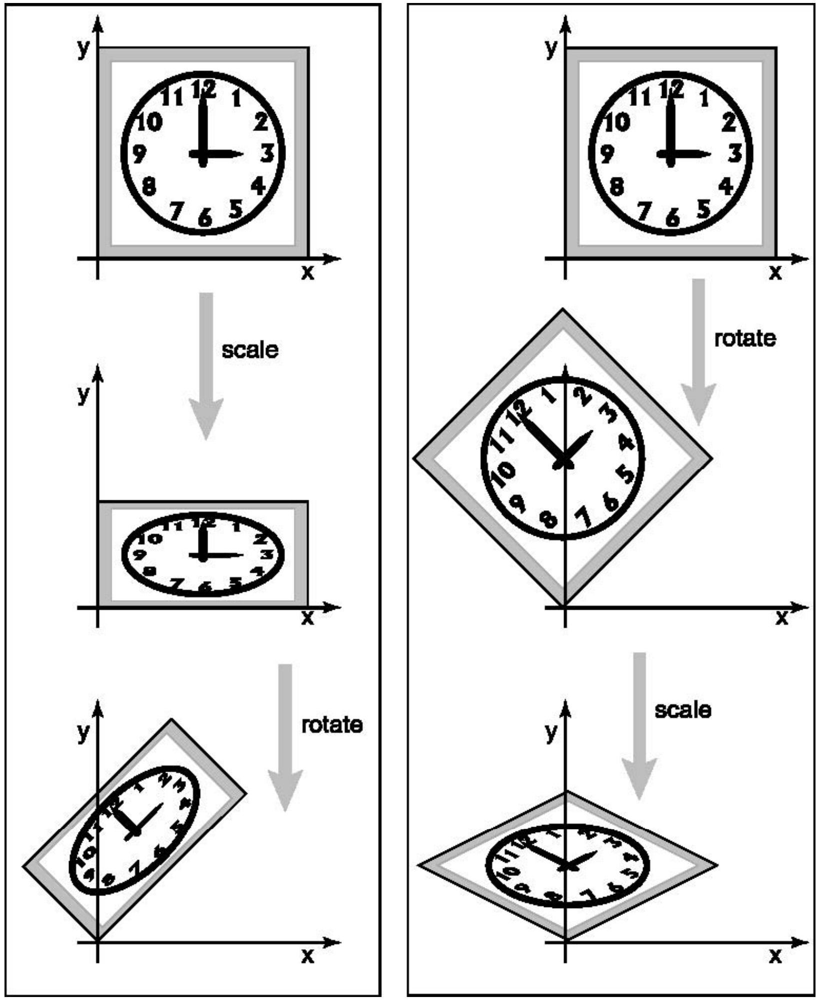
Composition of 2D Transformation

- Quite common to apply more than one transformations to an object
  - E.g., $v_2 = Sv_1$, $v_3 = Rv_2$, where S and R are scaling and Rotation matrix
- Then, we can use the following representation:
  - $v_3 = R(Sv_1)$ or
  - $v_3 = (RS)v_1$
  - why?



Transformation Order

- **Order of transforms is very important**
  - **Why?**



Affine Transformations

- **Transformed points (x', y') have the following form:**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

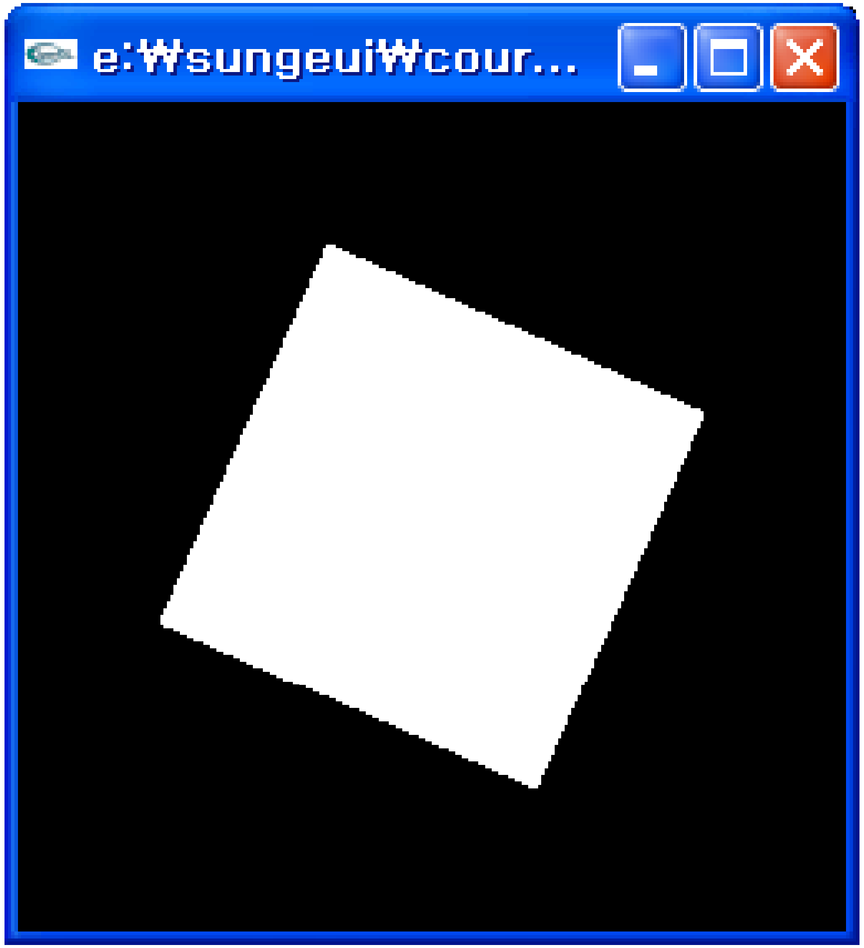
- **Combinations of translations, rotations, scaling, reflection, shears**
- **Properties**
  - **Parallel lines are preserved**
  - **Finite points map to finite points**

Rigid-Body Transforms in OpenGL

```
glTranslate (tx, ty, tz);
glRotate (angleInDegrees, axisX, axisY, axisZ);
glScale(sx, sy, sz);
```

OpenGL uses matrix format internally.

OpenGL Example – Rectangle Animation (double.c)



Demo

Main Display Function

```
void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);

  glPushMatrix();
  glRotatef(spin, 0.0, 0.0, 1.0);
  glColor3f(1.0, 1.0, 1.0);
  glRectf(-25.0, -25.0, 25.0, 25.0);
  glPopMatrix();

  glutSwapBuffers();
}
```

Frame Buffer

- Contains an image for the final visualization
- Color buffer, depth buffer, etc.

- Buffer initialization
  - glClear(GL_COLOR_BUFFER_BIT);
  - glClearColor (..);
- Buffer creation
  - glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
- Buffer swap
  - glutSwapBuffers();

Matrix Stacks

- OpenGL maintains matrix stacks
  - Provides pop and push operations
  - Convenient for transformation operations

- glMatrixMode() sets the current stack
  - GL_MODELVIEW, GL_PROJECTION, or GL_TEXTURE
- glPushMatrix() and glPopMatrix() are used to manipulate the stacks

OpenGL Matrix Operations

glTranslate(tx, ty, tz)

glRotate(angleInDegrees, axisX, axisY, axisZ)

glMultMatrix(*arrayOf16InColumnMajorOrder)

} Concatenate with the current matrix

glLoadMatrix (*arrayOf16InColumnMajorOrder)

glLoadIdentity()

} Overwrite the current matrix

Matrix Specification in OpenGL

- **Column-major ordering**

$$M = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

  - **Reverse to the typical C-convention (e.g., m [i][j] : row i & column j)**
  - **Better to declare m [16]**

- **Also, glLoadTransportMatrix*() & glMultTransposeMatrix*() are available**

Animation

- It consists of “redraw” and “swap”

- It’s desirable to provide more than 30 frames per second (fps) for interactive applications

- We will look at an animation example based on idle-callback function

Idle-based Animation

```
void mouse(int button, int state, int x, int y)
{
  switch (button) {
    case GLUT_LEFT_BUTTON:
      if (state == GLUT_DOWN)
        glutIdleFunc (spinDisplay);
      break;
    case GLUT_RIGHT_BUTTON:
      if (state == GLUT_DOWN)
        glutIdleFunc (NULL);
      break;
  }
}
```

```
void spinDisplay(void)
{
  spin = spin + 2.0;
  if (spin > 360.0)
    spin = spin - 360.0;
  glutPostRedisplay();
}
```

Class Objectives were:

- Write down simple 2D transformation matrixes
- Understand the homogeneous coordinates and its benefits
- Know OpenGL-transformation related API
- Implement idle-based animation method

Next Time

- 3D transformations