Viewing I: Model Transformations

Matrix Representation of Transformations

- Let $\mathcal{A}_0$ and $\mathcal{A}_1$ be affine spaces.
  Let $\mathbf{T} : \mathcal{A}_0 \mapsto \mathcal{A}_1$ be an affine transformation.
  Let $F_0 = (\vec{i}_0, \vec{j}_0, \mathcal{O}_0)$ be a frame for $\mathcal{A}_0$.
  Let $F_1 = (\vec{i}_1, \vec{j}_1, \mathcal{O}_1)$ be a frame for $\mathcal{A}_1$.
- Let $P = x\vec{i}_0 + y\vec{j}_0 + \mathcal{O}_0$ be a point in $\mathcal{A}_0$.
  The *coordinates* of P relative to $\mathcal{A}_0$ are $(x, y, 1)$.

  This can also be represented in vector form as $P = \begin{bmatrix} \vec{i}_0 & \vec{j}_0 & \mathcal{O}_0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

- What are the coordinates $(x', y', 1)$ of $\mathbf{T}(P)$ relative to F_1?
  - An affine transformation is characterized by the image of a frame in the domain.

$$\begin{aligned}\mathbf{T}(P) &= \mathbf{T}(x\vec{i}_0 + y\vec{j}_0 + \mathcal{O}_0) \\ &= x\mathbf{T}(\vec{i}_0) + y\mathbf{T}(\vec{j}_0) + \mathbf{T}(\mathcal{O}_0)\end{aligned}$$

  - $\mathbf{T}(\vec{i}_0)$ must be a linear combination of $\vec{i}_1$ and $\vec{j}_1$, say $\mathbf{T}(\vec{i}_0) = t_{1,1}\vec{i}_1 + t_{2,1}\vec{j}_1$.
  - Likewise $\mathbf{T}(\vec{j}_0)$ must be a linear combination of $\vec{i}_1$ and $\vec{j}_1$, say $\mathbf{T}(\vec{j}_0) = t_{1,2}\vec{i}_1 + t_{2,2}\vec{j}_1$.
  - Finally $\mathbf{T}(\mathcal{O}_0)$ must be an affine combination of $\vec{i}_1$, $\vec{j}_1$, and $\mathcal{O}_1$, say $\mathbf{T}(\mathcal{O}_0) = t_{1,3}\vec{i}_1 + t_{2,3}\vec{j}_1 + \mathcal{O}_1$.

– Then by substitution we get

$$\begin{aligned}
\mathbf{T}(P) &= x(t_{1,1}\vec{i}_1 + t_{2,1}\vec{j}_1) + y(t_{1,2}\vec{i}_1 + t_{2,2}\vec{j}_1) + t_{1,3}\vec{i}_1 + t_{2,3}\vec{j}_1 + \mathcal{O}_1 \\
&= \begin{bmatrix} t_{1,1}\vec{i}_1 + t_{2,1}\vec{j}_1 & t_{1,2}\vec{i}_1 + t_{2,2}\vec{j}_1 & t_{1,3}\vec{i}_1 + t_{2,3}\vec{j}_1 + \mathcal{O}_1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} \vec{i}_1 & \vec{j}_1 & \mathcal{O}_1 \end{bmatrix} \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\end{aligned}$$

Using $\mathbf{M}_T$ to denote the matrix, we see that $F_0 = F_1\mathbf{M}_T$

- Let $\mathbf{T}(P) = P' = x'\vec{i}_1 + y'\vec{j}_1 + \mathcal{O}_1$

In vector form this is

$$\begin{aligned} P' &= \begin{bmatrix} \vec{i}_1 & \vec{j}_1 & \mathcal{O}_1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \vec{i}_1 & \vec{j}_1 & \mathcal{O}_1 \end{bmatrix} \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \end{aligned}$$

So we see that

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

We can write this in shorthand – $\mathbf{p}' = \mathbf{M}_T\mathbf{p}$

- $\mathbf{M}_T$ is the *matrix representation* of $\mathbf{T}$
  - The first column of $\mathbf{M}_T$ represents $\mathbf{T}(\vec{i}_0)$
  - The second column of $\mathbf{M}_T$ represents $\mathbf{T}(\vec{j}_0)$
  - The third column of $\mathbf{M}_T$ represents $\mathbf{T}(\mathcal{O}_0)$

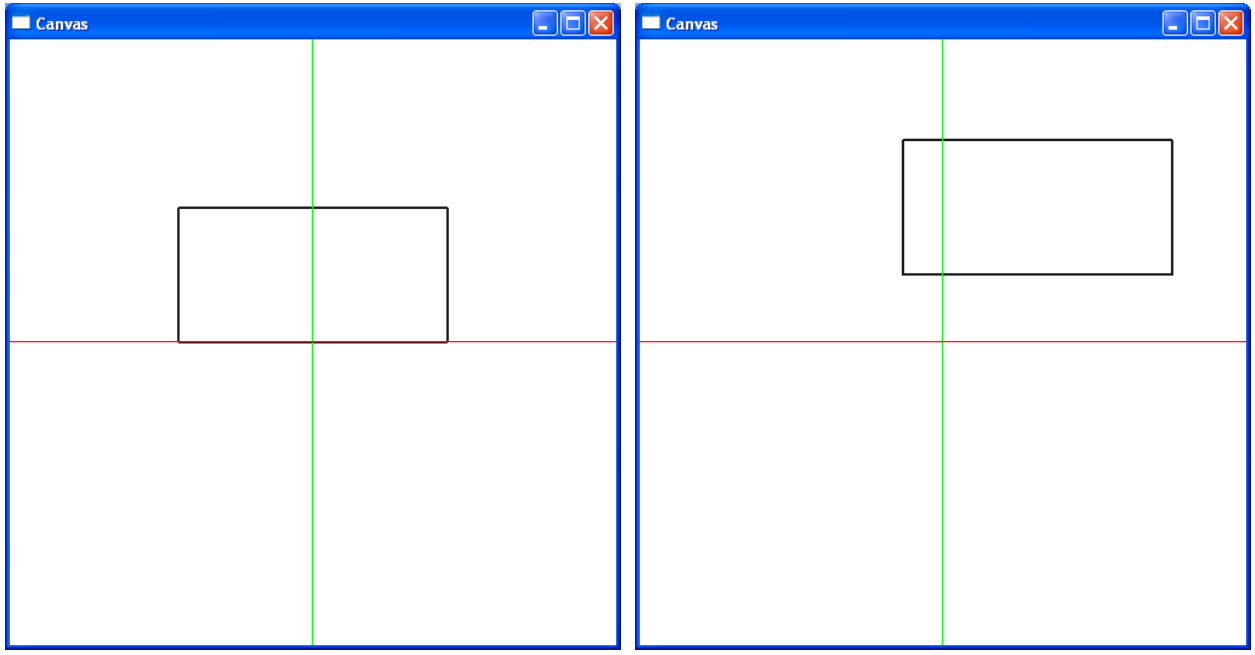
- *Translation*
  - Points are transformed as $\begin{bmatrix} x' & y' & 1 \end{bmatrix}^T = \begin{bmatrix} x & y & 1 \end{bmatrix}^T + \begin{bmatrix} \Delta x & \Delta y & 0 \end{bmatrix}^T$.
  - Vectors don't change.
  - Translation can be applied to sums of vectors and vector-point sums.
  - Matrix formulation:

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

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

  - Shorthand for the above matrix: $T(\Delta x, \Delta y)$

- *Example*




```
glTranslatef(.7, .5, 0);
glBegin(GL_LINE_LOOP);
    glVertex2f(-1, 0);
    glVertex2f(1, 0);
    glVertex2f(1,1);
    glVertex2f(-1,1);
glEnd();
```

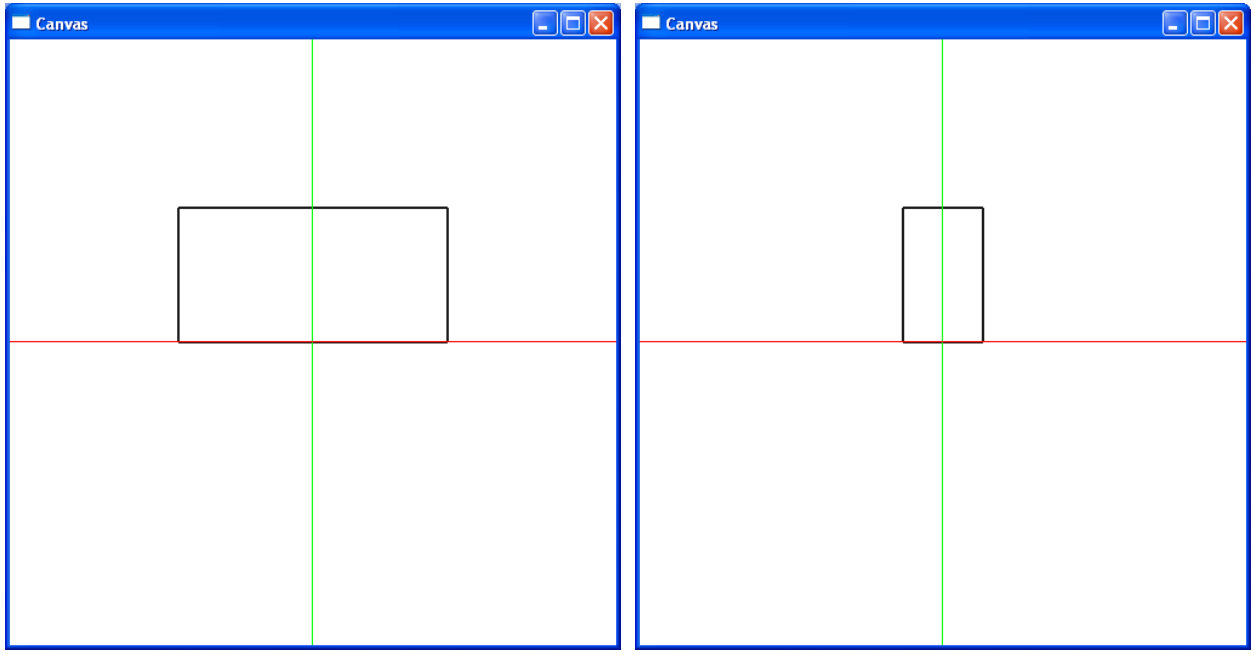
- *Scale*
  - Linear transform — applies equally to points and vectors
  - Points transform as $\begin{bmatrix} x' & y' & 1 \end{bmatrix}^T = \begin{bmatrix} xS_x & yS_y & 1 \end{bmatrix}^T$.
  - Vectors transform as $\begin{bmatrix} x' & y' & 0 \end{bmatrix}^T = \begin{bmatrix} xS_x & yS_y & 0 \end{bmatrix}^T$.
  - Matrix formulation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} xS_x \\ yS_y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} xS_x \\ yS_y \\ 0 \end{bmatrix}$$

  - Shorthand for the above matrix: $S(S_x, S_y)$
  - Note that this is *origin sensitive*.
  - How do you do reflections?

- *Example using OpenGL 2.5*




```
glScalef(0.3, 1, 1);
glBegin(GL_LINE_LOOP);
    glVertex2f(-1, 0);
    glVertex2f(1, 0);
    glVertex2f(1,1);
    glVertex2f(-1,1);
glEnd();
```

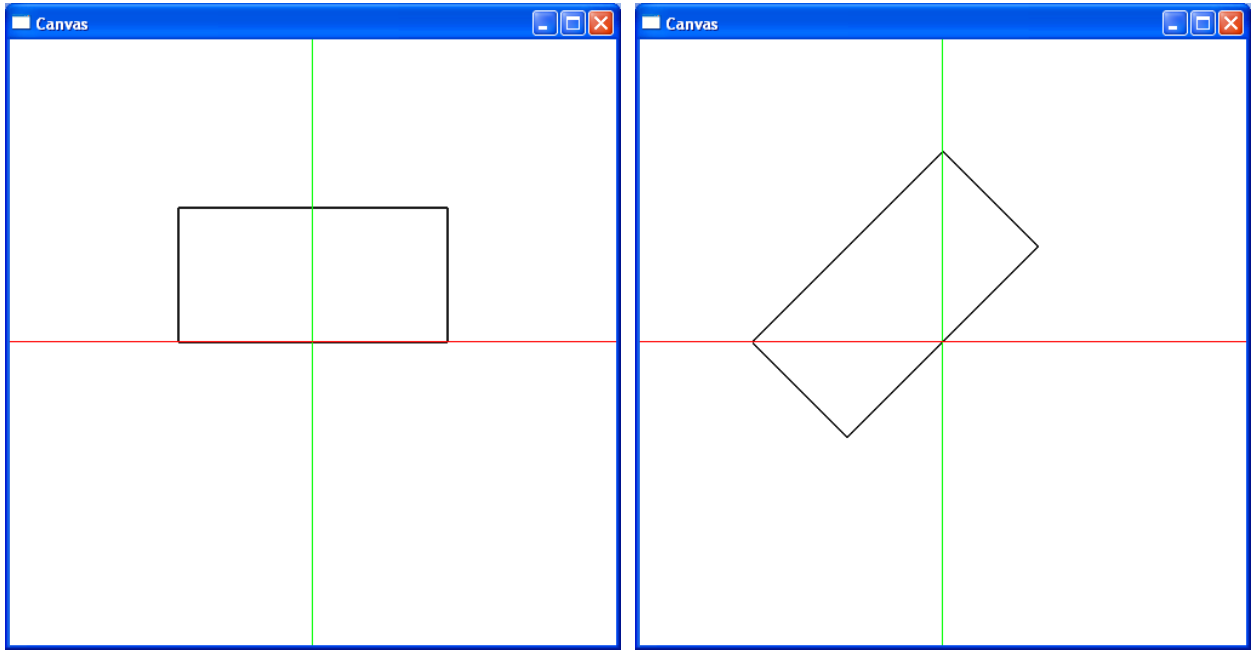
- *Rotate*
  - Linear transform — applies equally to points and vectors
  - Points transform as
    $\begin{bmatrix} x' & y' & 1 \end{bmatrix}^T = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) & x\sin(\theta) + y\cos(\theta) & 1 \end{bmatrix}^T$.
  - Vectors transform as
    $\begin{bmatrix} x' & y' & 0 \end{bmatrix}^T = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) & x\sin(\theta) + y\cos(\theta) & 0 \end{bmatrix}^T$.
  - Matrix formulation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \\ 0 \end{bmatrix}$$

  - Shorthand for the above matrix: $R(\theta)$
  - Note that this is *origin sensitive*.

- *Example*




```
glRotatef(45, 0, 0, 1);  /* camera by default is along z
glBegin(GL_LINE_LOOP);
    glVertex2f(-1, 0);
    glVertex2f(1, 0);
    glVertex2f(1,1);
    glVertex2f(-1,1);
glEnd();
```

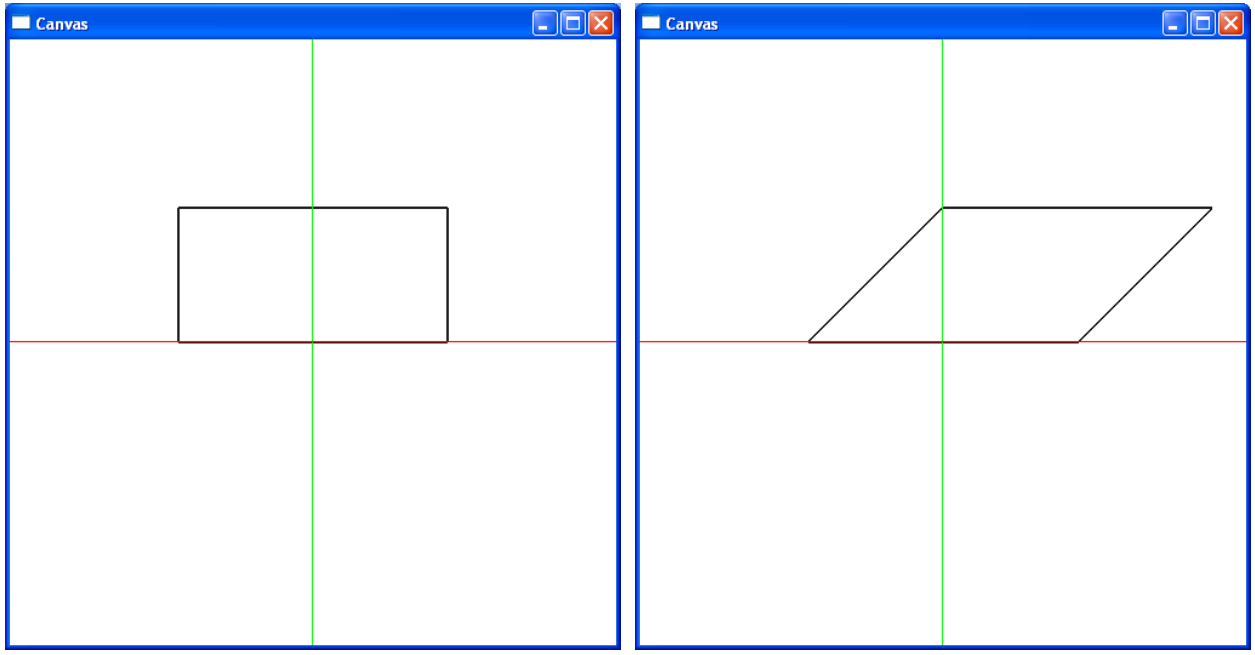
- *Shear*
  - Linear transform — applies equally to points and vectors
  - Points transform as $\begin{bmatrix} x' & y' & 1 \end{bmatrix}^T = [x + \alpha y,\ y + \beta x,\ 1]^T$.
  - Vectors transform as $\begin{bmatrix} x' & y' & 0 \end{bmatrix}^T = [x + \alpha y,\ y + \beta x,\ 0]^T$.
  - Matrix formulation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \alpha & 0 \\ \beta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + \alpha y \\ y + \beta x \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & \alpha & 0 \\ \beta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x + \alpha y \\ y + \beta x \\ 0 \end{bmatrix}$$

  - Shorthand for the above matrix: $Sh(\alpha, \beta)$

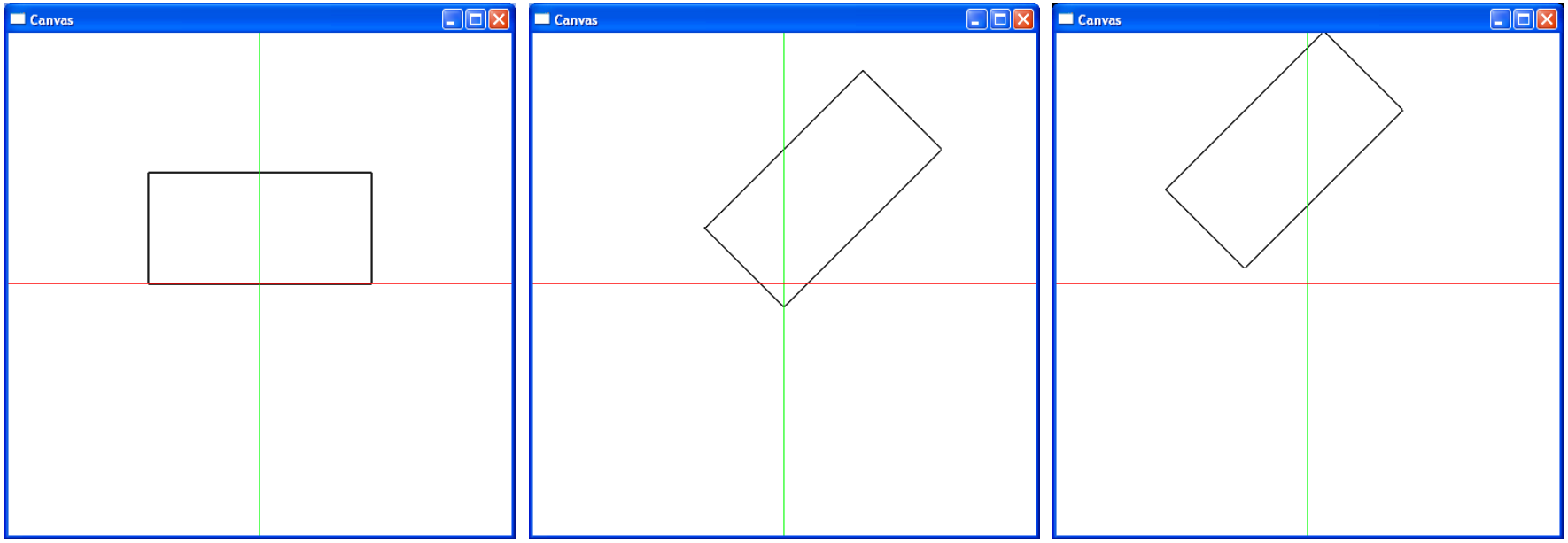
- *Example*




```
float ShearMatrix[] = {
        1, 1, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        0, 0, 0, 1  };
Traspose(ShearMatrix);
glMultMatrixf(ShearMatrix);
```

- Composition of Transformations
  - Now we have some basic transformations, how do we create and represent arbitrary affine transformations?
  - We can derive an arbitrary affine transform as a sequence of basic transformations, then compose the transformations
  - Example — scaling about an arbitrary point $[x_c\ y_c\ 1]^T$
    1. Translate $[x_c\ y_c\ 1]^T$ to $[0\ 0\ 1]$ $(T(-x_c, -y_c))$
    2. Scale $\left[x'\ y'\ 1\right]^T = S(S_x, S_y)\,[x\ y\ 1]^T$
    3. Translate $[0\ 0\ 1]^T$ back to $[x_c\ y_c\ 1]$ $(T(x_c, y_c))$
  - The sequence of transformation steps is
    $T(-x_c, -y_c) \circ S(S_x, S_y) \circ T(x_c, y_c)$

– *Example*




```
glTranslate(.7, .5, 0);
glRotatef(45,  0, 0, 1);
glBegin(GL_LINE_LOOP);
    glVertex2f(-1, 0);
    glVertex2f(1, 0);
    glVertex2f(1,1);
    glVertex2f(-1,1);
glEnd();
```

```
  glRotatef(45, 0, 0, 1);
glTranslate(.7, .5, 0);
  glBegin(GL_LINE_LOOP);
      glVertex2f(-1, 0);
      glVertex2f(1, 0);
      glVertex2f(1,1);
      glVertex2f(-1,1);
  glEnd();
```

– In matrix form this is

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_x & 0 & x_c(1-S_x) \\ 0 & S_y & y_c(1-S_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

– Note that the matrices are arranged from *right to left* in the order of the steps.
– The order is important (why)?

- Three Dimensional Transformations
  - A point is $\mathbf{p} = [x\ y\ z\ 1]$, a vector $\vec{v} = [x\ y\ z\ 0]$
  - Translation:

$$T(\Delta x, \Delta y, \Delta z) = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Scale:

$$S(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Rotation:

$$R_z(\Theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

    More on 3D Rotations later, especially using Quaternions!

OpenGL Transformation Matrices

There are three matrices that are part of the OpenGl pipeline, and all are manipulated by a common set of functions. To select the matrix type on which operations apply use glMatrixMode function. For example,

```
glMatrixMode(GL_MODELVIEW); or  glMatrixMode(GL_PROJECTION)
```

- The matrix applied to all primitives is the product of the ModelView matrix and the Projection matrix.
- Matrix is loaded with function

  `glLoadMatrixfv(pointer_to_matrix)`
- Matrix is altered with function

  `glMultMatrixfv(pointer_to_matrix)`
- Translation is provided with function

  `glTranslatef(dx,dy,dz)`
- Rotation is provided with function

  `glRotatef(angle,vx,vy,vz)`

- Scaling is provided with function

  `glScalef(sx,sy,sz)`

- All three transformations alter the selected matrix by postmultiplication.

Order of Applying Transformations The rule in OpenGL: The transformation specified last is the one applied first.

Consider the example sequence to form the required matrix for a 45-degree rotation about a vector (1,2,3). The object frame's origin is (4,5,6) and that is its center of rotation. The sequence is to move the object's frame to the origin (0,0,0), rotating about the origin, and finally moving the rotated object back to its original location.

```
glMatrixModel(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(4.0,5.0,6.0);
glRotatef(45.0, 1.0,2.0,3.0);
glTranslatef(-4.0,-5.0,-6.0);
```

Reading Assignment and News

Before the next class please review Chapter 3 and its practice exercises, of the recommended text.

(Recommended Text: Interactive Computer Graphics, by Edward Angel, Dave Shreiner, 6th edition, Addison-Wesley)

Please track Blackboard for the most recent Announcements and Project postings related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics2012/cs354/)