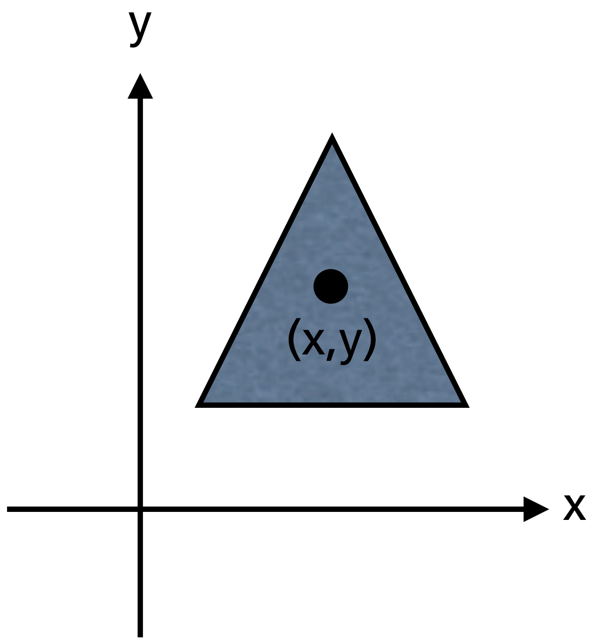


Original position of object and pivot point

Monash University • Clayton's School of Information Technology

CSE3313 Computer Graphics

Lecture 10: Homogeneous Transformations in 2D

Homogeneous Coordinates

We have the homogeneous representations for translation, rotation and scaling:

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

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

$$\mathbf{S}(\alpha, \beta) = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- If we use an affine transformation to produce a transformed point $p' = \mathbf{A}\, p$, then if $\mathbf{A}^{-1}$ exists we have $p = \mathbf{A}^{-1}\, p'$.
- $\mathbf{A}^{-1}$ is the inverse transformation, that is, the transformation that undoes the effect of $\mathbf{A}$.

Inverse Transforms

- For the basic transformations of translation, rotation and scaling the inverse transformations are easy to calculate:

$$\mathbf{T}^{-1}(\Delta x, \Delta y) = \mathbf{T}(-\Delta x, -\Delta y)$$

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$$

$$\mathbf{S}^{-1}(\alpha, \beta) = \mathbf{S}(\frac{1}{\alpha}, \frac{1}{\beta})$$

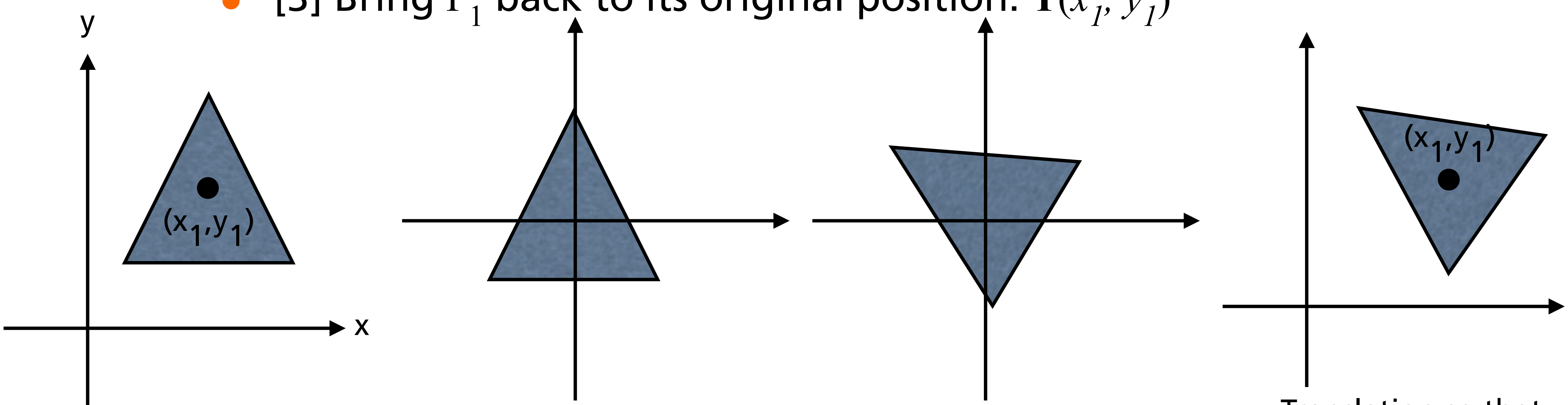
Note: if α or β are zero then everything gets squashed onto a line or a point and the inverse transformation would not exist.

Furthermore if a composite transformation is given by $\mathbf{A}\,\mathbf{B}$, transform by $\mathbf{B}$, followed by a transform by A then:

$(\mathbf{A}\,\mathbf{B})^{-1} = \mathbf{B}^{-1}\,\mathbf{A}^{-1}$, undo transformation by $\mathbf{A}$, then undo transformation by $\mathbf{B}$.

Concatenated Transforms

- Rotation about a point $P_1 = (x_1, y_1)$
- The basic transformation is to rotate about the origin.
- To rotate about an arbitrary point:
  - [1] Bring P_1 to the origin: $\mathbf{T}(-x_1, -y_1)$
  - [2] Rotate by θ about the origin: $\mathbf{R}(\theta)$
  - [3] Bring P_1 back to its original position: $\mathbf{T}(x_1, y_1)$



Original position of object and pivot point

Translation of pivot point to origin.

Rotation about origin.

Translation so that pivot point is returned to its original position.

Rotation about a point (cont.)

- The overall transformation is:

$$
\begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} \cos\theta & -\sin\theta & -x_1 \cos + y_1 \sin\theta \\ \sin\theta & \cos\theta & -x_1 \sin\theta - y_1 \cos\theta \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} \cos\theta & -\sin\theta & -x_1 \cos + y_1 \sin\theta + x_1 \\ \sin\theta & \cos\theta & -x_1 \sin\theta - y_1 \cos\theta + y_1 \\ 0 & 0 & 1 \end{bmatrix}
$$

Compound Transforms: Viewport Transform

- The viewport transformation:

$$wx_{\min}, wx_{\max}, wy_{\min}, wy_{\max}$$

defines a window in world coordinates while

$$vx_{\min}, vx_{\max}, vy_{\min}, vy_{\max}$$

defines a viewport in normalised device coordinates.

Let us define:

$$s_x = \frac{vx_{\max} - vx_{\min}}{wx_{\max} - wx_{\min}}, \quad s_y = \frac{vy_{\max} - vy_{\min}}{wy_{\max} - wy_{\min}}$$

s_x is the scaling in the x direction. s_y is the scaling in the y direction.

Viewport Transformation (cont.)

- The viewport transformation can be expressed as:
  - Translate (wx_{min}, wy_{min}) to the origin;
  - Scale by s_x and s_y;
  - Translate origin to (vx_{min}, vy_{min}).

$$\begin{bmatrix} 1 & 0 & vx_{\min} \\ 0 & 1 & vy_{\min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -wx_{\min} \\ 0 & 1 & -wy_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & vx_{\min} \\ 0 & 1 & vy_{\min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & -s_x wx_{\min} \\ 0 & s_y & -s_y wy_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x & 0 & vx_{\min} - s_x wx_{\min} \\ 0 & s_y & vy_{\min} - s_y wy_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

Combining Transformations

- There may be more than one way of combining basic transformations to achieve a required complex transformation.
- For example the viewport transformation could be expressed as
  - [1] translate the centre of the window to the origin;
  - [2] scale so that the window and the viewport are the same size;
  - [3] translate the origin to the centre of the viewport.
- Note that all these transformations result in a 3 x 3 matrix whose last row is:

  $$[\,0 \quad 0 \quad 1\,]$$

- Normally, multiplying a 3 x 3 matrix by a 3 x 1 vector costs 9 multiplications.
- For transformation matrices only 4 multiplications are required when the scaling factor in homogeneous coordinates is set to 1.
- These facts might be used to implement transformations efficiently, even if it is more convenient to treat them conceptually as 3 x 3 matrices.

Transformations in OpenGL

- The order in which transformations is applied matters. Changing that order may lead to a different composite transform.
- Transformations applied to objects prior to the viewport transformation are called **object** or **modelling** transformations.
- Transformations applied to objects after the viewport transformation are called image transformations.
- OpenGL has transformation matrices that are part of the state of the graphics system. The two most important are the **model-view** and the **projection** matrices. Both matrices start off as **identity matrices.**
- The model-view matrix converts world coordinates to viewing coordinates, i.e. coordinates relative to the viewer or synthetic camera.

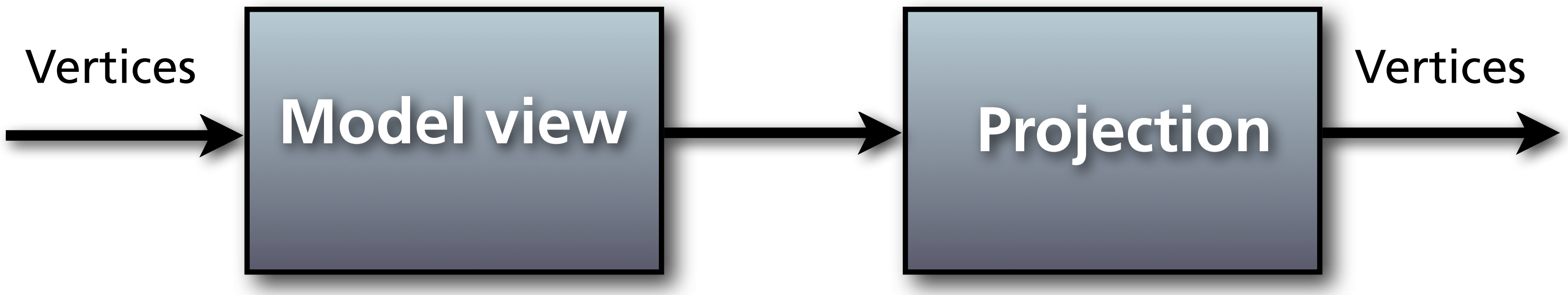
Transformations in OpenGL (cont.)

- The projection matrix is used to transform the viewing coordinates of objects to 2D device coordinates.
- Operations in OpenGL are applied to the current matrix only. The current matrix is chosen by setting the **matrix mode**. The default mode is `GL_MODELVIEW`.
- For example, in sample programs we might have the following code in an initialization routine like `myInit`.

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity( );
gluOrtho2D( 0.0, 500.0, 0.0, 500.0 );
glMatrixMode( GL_MODELVIEW );
```

  This code follows the convention of always leaving the matrix mode in a default state — in this case `GL_MODELVIEW`

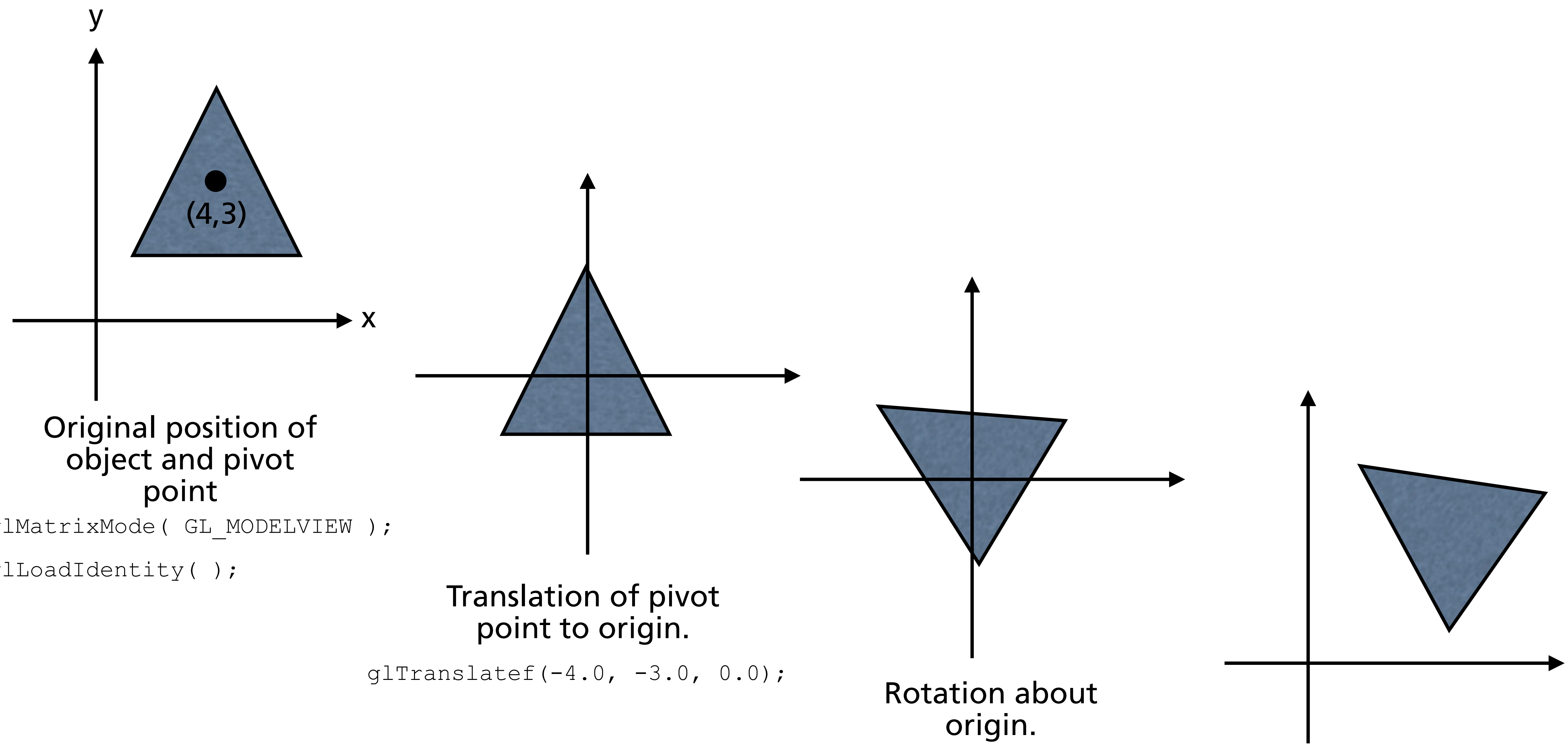
Transformations in OpenGL (cont)



- OpenGL Transformations
  - `glRotatef(angle, vx, vy, vz)` — rotate about the vector (vx, vy, vz) by `angle` degrees. (see also `glRotated` — double prec. version).
  - `glTranslatef(dx, dy, dz)` – translation $\mathbf{T}(dx, dy, dz)$
  - `glScale(sx, sy, sz)` – scale $\mathbf{S}(sx, sy, sz)$
- All these transformation routines alter the selected matrix by post-multiplication.

Transformations in OpenGL

- How to do this compound transformation in OpenGL?



Original position of object and pivot point

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );
```

Translation of pivot point to origin.

```
glTranslatef(-4.0, -3.0, 0.0);
```

Rotation about origin.

```
glRotatef(55.0, 0.0, 0.0, 1.0);
```

Translation so that pivot point is returned to its original position.

```
glTranslatef(4.0, 3.0, 0.0);
```

Compound Transformation Example

Reference: Angel Section 4.9

- Rotation about a fixed point:

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );
glTranslatef(4.0, 3.0, 0.0);
glRotatef(55.0, 0.0, 0.0, 1.0);
glTranslatef(-4.0, -3.0, 0.0);
```

- Order of transformations:

$$\mathbf{C} \leftarrow \mathbf{I}$$

$$\mathbf{C} \leftarrow \mathbf{C}\ \mathbf{T}(4.0, 3.0, 0.0)$$

$$\mathbf{C} \leftarrow \mathbf{C}\ \mathbf{R}(55.0, 0.0, 0.0, 1.0)$$

$$\mathbf{C} \leftarrow \mathbf{C}\ \mathbf{T}(-4.0, -3.0, 0.0)$$

- Each vertex, $\mathbf{p}$, that is sent *after* the model-view matrix has been set will be multiplied by $\mathbf{C}$, thus forming a new vertex:

$$\mathbf{p'} = \mathbf{C}\ \mathbf{p}$$