Geometry: 2-D Transformations

Outline

- Effects
- Mathematical representation
- OpenGL functions for applying
Why We Need Transformations

- Objects may have different locations, scales, and orientations
- Complex objects may be constructed by the transformation of simple objects
- Camera may have different locations and orientations

Example: Shape vs. Viewing Issues
Example: Shape vs. Viewing Issues

Example: Shape vs. Viewing Issues
Example: Shape vs. Viewing Issues

Transformations for modeling

1. Building complex objects from simpler parts
2-D Transformations

• Types
  - Translation
  - Scaling
  - Rotation
  - Shear, reflection

2-D Translation
2-D Translation

\[
\begin{pmatrix} x \\ y \end{pmatrix}
\]
2-D Translation

\[
\begin{pmatrix}
  x \\
y
\end{pmatrix} + \begin{pmatrix}
  \Delta x \\
  \Delta y
\end{pmatrix}
\]

2-D Translation

\[
\begin{pmatrix}
  x' \\
y'
\end{pmatrix} = \begin{pmatrix}
  x \\
y
\end{pmatrix} + \begin{pmatrix}
  \Delta x \\
  \Delta y
\end{pmatrix}
\]
2-D Scaling

Horizontal shift proportional to horizontal position

2-D Scaling

Vertical shift proportional to vertical position
2-D Scaling

\[
\begin{pmatrix}
  x' \\
  y'
\end{pmatrix} =
\begin{pmatrix}
  s_x \cdot x \\
  s_y \cdot y
\end{pmatrix}
\]
2-D Rotation

This is a counterclockwise rotation
2-D Rotation

This is a counterclockwise rotation

\[
\begin{pmatrix}
  x' \\
y'
\end{pmatrix} =
\begin{pmatrix}
x \cos \theta - y \sin \theta \\
x \sin \theta + y \cos \theta
\end{pmatrix}
\]
2-D Rotation

\[
\begin{pmatrix}
  x' \\
  y'
\end{pmatrix}
= \begin{pmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta
\end{pmatrix}
\begin{pmatrix}
  x \\
  y
\end{pmatrix}
\]

2-D Rotation (uncentered)

\[
\begin{pmatrix}
  x' \\
  y'
\end{pmatrix}
= \begin{pmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta
\end{pmatrix}
\begin{pmatrix}
  x \\
  y
\end{pmatrix}
\]
2-D Rotation (uncentered)

\[
\begin{pmatrix}
x' \\
y'
\end{pmatrix} =
\begin{pmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{pmatrix}
\begin{pmatrix}
x \\
y
\end{pmatrix}
\]

2-D Shear (horizontal)
2-D Shear (horizontal)

Horizontal displacement proportional to vertical position

\[
\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\]
2-D Reflection (vertical)

Just a special case of scaling—"negative" scaling
Representing Transformations

- Note that we've defined translation as a vector addition but rotation, scaling, etc. as matrix multiplications.
- It's inconvenient to have two different operations (addition and multiplication) for different forms of transformation.
- It would be desirable for all transformations to be expressed in a common form.

\[
\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\]
Representing Transformations

• Note that we've defined translation as a vector addition but rotation, scaling, etc. as matrix multiplications
• It’s inconvenient to have two different operations (addition and multiplication) for different forms of transformation
• It would be desirable for all transformations to be expressed in a common form
  - Solution: Homogeneous coordinates

Homogeneous Coordinates

• Let \( \mathbf{x} = (x_1, \ldots, x_n)^T \) be a point in Euclidean space
• Change to homogeneous coordinates:
  \[
  \mathbf{x} \Rightarrow (\mathbf{x}^T, 1)^T
  \]
• Defined up to scale
  \[
  (\mathbf{x}^T, 1)^T \Rightarrow (w\mathbf{x}^T, w)^T
  \]
• Can go back to non-homogeneous representation as follows:
  \[
  (\mathbf{x}^T, w)^T \Rightarrow \mathbf{x}/w
  \]
Homogeneous Coordinates: Translations

- 2-D translation of a point was expressed as a vector addition $\mathbf{x}^0 = \mathbf{x} + \mathbf{t}$
- Homogeneous coordinates allow it to be written as a multiplication by a $3 \times 3$ matrix:

$$
\mathbf{x}' = \begin{pmatrix}
\text{Id} & \mathbf{t} \\
0^T & 1
\end{pmatrix} \mathbf{x}
$$

Example: Translation with homogeneous coordinates

- Old way: 
  $$
  \begin{pmatrix}
  x' \\
y'
  \end{pmatrix} = \begin{pmatrix}
x \\
y
\end{pmatrix} + \begin{pmatrix}
\Delta x \\
\Delta y
\end{pmatrix}
  $$
- New way:
  $$
  \begin{pmatrix}
x' \\
y' \\
1
\end{pmatrix} = \begin{pmatrix}
1 & 0 & \Delta x \\
0 & 1 & \Delta y \\
0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
  $$
Homogeneous Coordinates: Rotations, etc.

- A 2-D rotation, scaling, shear or other transformation normally expressed by a $2 \times 2$ matrix $R$ is written in homogeneous coordinates with the following $3 \times 3$ matrix:

$$x' = \begin{pmatrix} R & 0 \\ 0^T & 1 \end{pmatrix} x$$

- The non-commutativity of matrix multiplication explains why different transformation orders give different results—i.e., $RT \neq TR$

2-D Transformations: Tilted Axes

- All of the scalings, reflections, etc. described so far are relative to the coordinate axes
- How can we perform a transformation relative to some tilted axis?
- Basic idea:
  1. Apply rotation so that tilted axis is aligned with a coordinate axis
  2. Apply desired transformation (reflection, shear, etc.) for that coordinate axis
  3. Apply inverse rotation so that tilted axis is “restored”
OpenGL’s coordinates

- The underlying form of all points/vertices is a 4-D vector \((x, y, z, w)\)
- If you do something in 2-D, OpenGL simply sets \(z = 0\) for you
- If the scale coordinate \(w\) is not set explicitly, OpenGL sets \(z = 1\) for you

2-D Transformations: OpenGL

- 2-D transformation functions
  - `glTranslate(x, y, 0)`
  - `glScale(sx, sy, 0)`
  - `glRotate(theta, 0, 0, 1)` (angle in degrees; direction is counterclockwise)
- Notes
  - Set `glMatrixMode(GL_MODELVIEW)` first
  - Transformations should be specified before drawing commands to be affected
  - Multiple transformations are applied in reverse order
Example: 2-D Translation in OpenGL

Limiting “Scope” of Transformations

- Transformations are ordinarily applied to all subsequent draw commands.
- To limit effects, use push/pop functions:
  ```
  glPushMatrix();
  // transform
  // draw affected by transform
  glPopMatrix();
  // draw unaffected by transform
  ```