Transformations

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

(with lots of help from Prof. Emmanuel Agu :-)

CS 543: Computer Graphics
Introduction to Transformations

- A transformation changes an object's
  - Size (scaling)
  - Position (translation)
  - Orientation (rotation)
  - Shape (shear)

- We will introduce first in 2D or \((x,y)\), build intuition

- Later, talk about 3D and 4D?

- Transform object by applying sequence of matrix multiplications to object vertices
Why Matrices?

- All transformations can be performed using matrix/vector multiplication
- Allows pre-multiplication of all matrices
- Note: point \((x, y)\) needs to be represented as \((x, y, 1)\), also called **homogeneous coordinates**
Point Representation

- We use a column matrix (2x1 matrix) to represent a 2D point

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

- General form of transformation of a point \((x, y)\) to \((x', y')\) can be written as:

\[
x' = ax + by + c \quad \text{or} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}
\]

\[
y' = dx + ey + f
\]
Translation

To reposition a point along a straight line

Given point \((x, y)\) and translation distance \((t_x, t_y)\)

The new point: \((x', y')\)

\[
x' = x + t_x
\]
\[
y' = y + t_y
\]

or

\[
P' = P + T
\]

where

\[
P' = \begin{pmatrix} x' \\ y' \end{pmatrix}
\]
\[
P = \begin{pmatrix} x \\ y \end{pmatrix}
\]
\[
T = \begin{pmatrix} t_x \\ t_y \end{pmatrix}
\]
3x3 2D Translation Matrix

\[ x' = x + t_x \]
\[ y' = y + t_y \]

\[
\begin{pmatrix}
  x' \\
  y' \\
  1
\end{pmatrix}
= \begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix} + \begin{pmatrix}
  t_x \\
  t_y \\
  0
\end{pmatrix}
\]

use 3x1 vector

\[
\begin{pmatrix}
  x' \\
  y' \\
  1
\end{pmatrix}
= \begin{pmatrix}
  1 & 0 & t_x \\
  0 & 1 & t_y \\
  0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
\]

Note: it becomes a matrix-vector multiplication
Translation of Objects

- How to translate an object with multiple vertices?

Translate individual vertices
2D Scaling

- Scale: Alter object size by scaling factor \((sx, sy)\). *i.e.*, 

\[
x' = x \times Sx \\
y' = y \times Sy
\]

\[
\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\]
3x3 2D Scaling Matrix

\[ x' = x \times S_x \]
\[ y' = y \times S_y \]

\[
\begin{pmatrix}
  x' \\
  y' \\
  1
\end{pmatrix}
= \begin{pmatrix}
  S_x & 0 & 0 \\
  0 & S_y & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
\]
Shearing

- Y coordinates are unaffected, but x coordinates are translated linearly with y.
- That is
  \[ x' = x + y \times h \]
  \[ y' = y \]
- h is fraction of y to be added to x.

\[
\begin{pmatrix}
 x' \\
 y' \\
 1
\end{pmatrix} =
\begin{pmatrix}
 1 & h & 0 \\
 0 & 1 & 0 \\
 0 & 0 & 1
\end{pmatrix}
\times
\begin{pmatrix}
 x \\
 y \\
 1
\end{pmatrix}
\]
2D Rotation

Default rotation center is origin \((0,0)\)

\[\theta > 0 : \text{Rotate counter clockwise}\]

\[\theta < 0 : \text{Rotate clockwise}\]
2D Rotation (cont.)

\[(x, y) \rightarrow \text{Rotate about the origin by } \theta\]

How to compute \((x', y')\)?

\[
x = r \cdot \cos(\phi) \quad x' = r \cdot \cos(\phi + \theta)
\]
\[
y = r \cdot \sin(\phi) \quad y' = r \cdot \sin(\phi + \theta)
\]
2D Rotation (cont.)

Using trig. identities

\[
\cos(\theta + \phi) = \cos \theta \cos \phi - \sin \theta \sin \phi \\
\sin(\theta + \phi) = \sin \theta \cos \phi + \cos \theta \sin \phi
\]

\[
x' = x \cos(\theta) - y \sin(\theta) \\
y' = x \sin(\theta) + y \cos(\theta)
\]

Matrix form?

\[
\begin{pmatrix}
x' \\
y'
\end{pmatrix} = \begin{pmatrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{pmatrix} \begin{pmatrix}
x \\
y
\end{pmatrix}
\]
**3x3 2D Rotation Matrix**

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

\[
\begin{pmatrix}
  x' \\
  y'
\end{pmatrix} =
\begin{pmatrix}
  \cos(\theta) & -\sin(\theta) & 0 \\
  \sin(\theta) & \cos(\theta) & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
\]
2D Rotation

How to rotate an object with multiple vertices?

Rotate individual Vertices

\[ \theta \]
Arbitrary Rotation Center

- To rotate about arbitrary point \( P = (Px, Py) \) by \( \theta \):
  - Translate object by \( T(-Px, -Py) \) so that \( P \) coincides with origin
  - Rotate the object by \( R(\theta) \)
  - Translate object back: \( T(Px, Py) \)

- In matrix form
  - \( T(Px,Py) \) \( R(\theta) \) \( T(-Px,-Py) \) \(*\) \( P \)

\[
\begin{pmatrix}
  x' \\
  y' \\
  1
\end{pmatrix} = \begin{pmatrix}
  1 & 0 & Px \\
  0 & 1 & Py \\
  0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
  \cos(\theta) & -\sin(\theta) & 0 \\
  \sin(\theta) & \cos(\theta) & 0 \\
  0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
  1 & 0 & -Px \\
  0 & 1 & -Py \\
  0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
\]

- Similar for arbitrary scaling anchor
Composing Transformations

- Composing transformations
  - Applying several transforms in succession to form one overall transformation

- Example
  - $M_1 \times M_2 \times M_3 \times P$
    - where $M_1$, $M_2$, $M_3$ are transform matrices applied to $P$

- Be careful with the order!

- For example
  - Translate by $(5, 0)$, then rotate 60 degrees is NOT same as
  - Rotate by 60 degrees, then translate by $(5, 0)$
OpenGL Transformations

- Designed for 3D
- For 2D, simply ignore z dimension
- Translation:
  - \( \text{glTranslated}( tx, ty, tz ) \)
  - \( \text{glTranslated}( tx, ty, 0 ) \) => for 2D
- Rotation:
  - \( \text{glRotated}( \text{angle}, Vx, Vy, Vz ) \)
  - \( \text{glRotated}( \text{angle}, 0, 0, 1 ) \) => for 2D
- Scaling:
  - \( \text{glScaled}( sx, sy, sz ) \)
  - \( \text{glScaled}( sx, sy, 0 ) \) => for 2D
3D Transformations

- **Affine transformations**
  - Mappings of points to new points that retain certain relationships
  - Lines remain lines
  - Several transformations can be combined into a single matrix

- **Two ways to think about transformations**
  - Object transformations
    - All points of an object are transformed
  - Coordinate transformations
    - The coordinate system is transformed, and models remain defined relative to this
3D Transformations (cont.)

- **Scale**
  - `glScaled( sx, sy, sz )`: Scale object by (sx, sy, sz)

- **Translate**
  - `glTranslated( dx, dy, dz )`: Translate object by (dx, dy, dz)

- **Rotate**
  - `glRotated( angle, ux, uy, uz )`: Rotate by angle about an axis passing through origin and (ux, uy, uz)

- **OpenGL**
  - Creates a matrix for each transformation
  - Multiplies matrices together to form a single, combined matrix
  - Transformation matrix is called *modelview matrix*
Example: Translation

- The vertices of an object are mapped to new points
  - Similarly for scaling and rotation
OpenGL Matrices

- Graphics pipeline takes all vertices through a series of operations

```
VM → P → clip → V_p

projection matrix

modelview matrix

viewport matrix
```
OpenGL Matrices and the Pipeline

- OpenGL uses three matrices for geometry
  - Modelview matrix
  - Projection matrix
  - Viewport matrix

- Modelview matrix
  - Combination of modeling matrix \( M \) and camera transforms \( V \)

- Other OpenGL matrices include texture and color matrices

- \texttt{glMatrixMode} command selects matrix mode

- \texttt{glMatrixMode} parameters
  - \texttt{GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE}, etc.

- May initialize matrices with \texttt{glLoadIdentity()}
OpenGL Matrices and the Pipeline

- OpenGL matrix operations are 4x4 matrices
- Graphics card
  - Fast 4x4 multiplier -> tremendous speedup
View Frustum

- Side walls determined by window borders
- Other walls are programmer defined
  - Near clipping plane
  - Far clipping plane
- Transform 3D models to 2D
  - Project points/vertices inside view volume onto view window using parallel lines along z-axis
Types of Projections

- Different types of projections?
  - Different view volume shapes
  - Different visual effects

- Example projections
  - Parallel (a.k.a. orthographic)
  - Perspective

- Parallel is simple

- Will use this for intro, expand later
OpenGL Matrices and the Pipeline

- Projection matrix
  - Scales and shifts each vertex in a particular way
  - View volume lies inside cube of –1 to 1
  - Reverses sense of z
    - increasing z = increasing depth
  - Effectively squishes view volume down to cube centered at 1

- Clipping in 3D then eliminates portions outside view frustum

- Viewport matrix:
  - Maps surviving portion of block (cube) into a 3D viewport
  - Retains a measure of the depth of a point