2D Transformations

- Given a 2D object, transformation is to change the object’s:
  - Position (translation)
  - Size (scaling)
  - Orientation (rotation)
  - Shapes (shear)
- Apply a sequence of matrix multiplication to the object vertices

2D Transformation

- We can use a column vector (a 2x1 matrix) to represent a 2D point:
  \[
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  \]
- A general form of linear transformation can be written as:
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  a & b & c \\
  d & e & f
  \end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  1
  \end{pmatrix}
  \]
- Translation
  - Re-position a point along a straight line
  - Given a point \((x,y)\), and the translation distance \((tx,ty)\)
  - The new point: \((x',y')\)
    \[
    x' = x + tx \\
    y' = y + ty
    \]
    \[
    \begin{pmatrix}
    x' \\
    y'
    \end{pmatrix} = \begin{pmatrix}
    1 & 0 & tx \\
    0 & 1 & ty
    \end{pmatrix} \begin{pmatrix}
    x \\
    y \\
    1
    \end{pmatrix}
    \]
3x3 2D Translation Matrix

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  x & tx \\
  y & ty
\end{bmatrix}
\]

Use 3 x 1 vector

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & tx \\
  0 & 1 & ty
\end{bmatrix} \cdot \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

• Note that now it becomes a matrix-vector multiplication

Translation

• How to translate an object with multiple vertices?

2D Rotation

• Default rotation center: Origin (0,0)

θ > 0 : Rotate counter clockwise

θ < 0 : Rotate clockwise

\[
x = r \cos (\phi) \\
y = r \sin (\phi)
\]

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

Rotation

(x, y) -> Rotate about the origin by θ

How to compute (x', y')?

x = r \cos (\phi) \\
y = r \sin (\phi)

x' = r \cos (\phi + \theta) \\
y' = r \sin (\phi + \theta)
Rotation

\[
x = r \cos(\phi) \quad y = r \sin(\phi) \\
x' = r \cos(\phi + \theta) \quad y = r \sin(\phi + \theta)
\]

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

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

3x3 2D Rotation Matrix

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

3 x 3? 

How to rotate an object with multiple vertices?

Rotate individual Vertices
2D Scaling

Scale: Alter the size of an object by a scaling factor $(S_x, S_y)$, i.e.

$$x' = x \cdot S_x$$
$$y' = y \cdot S_y$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Not only the object size is changed, it also moved!!
Usually this is an undesirable effect
We will discuss later (soon) how to fix it

3x3 2D Scaling Matrix

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Put it all together

- Translation: $$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$
- Rotation: $$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$
- Scaling: $$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$
Or, 3x3 Matrix representations

- Translation:
  \[
  \begin{bmatrix}
  x' \\
y' \\
1
  \end{bmatrix} =
  \begin{bmatrix}
  1 & 0 & tx \\
0 & 1 & ty \\
0 & 0 & 1
  \end{bmatrix}
  \begin{bmatrix}
  x \\
y \\
1
  \end{bmatrix}
  \]

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

- Scaling:
  \[
  \begin{bmatrix}
  x' \\
y' \\
1
  \end{bmatrix} =
  \begin{bmatrix}
  Sx & 0 & 0 \\
0 & Sy & 0 \\
0 & 0 & 1
  \end{bmatrix}
  \begin{bmatrix}
  x \\
y \\
1
  \end{bmatrix}
  \]

Why use 3x3 matrices?

- So that we can perform all transformations using matrix/vector multiplications
- This allows us to pre-multiply all the matrices together
- The point (x,y) needs to be represented as (x,y,1) - this is called Homogeneous coordinates!

Shearing

- Y coordinates are unaffected, but x coordinates are translated linearly with y
- That is:
  - \(y' = y\)
  - \(x' = x + y \cdot h\)

Shearing in y

Interesting Facts:
- A 2D rotation is three shears
- Shearing will not change the area of the object
- Any 2D shearing can be done by a rotation, followed by a scaling, and followed by a rotation
**Rotation Revisit**

- The standard rotation matrix is used to rotate about the origin (0,0):

\[
\begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

- What if I want to rotate about an arbitrary center?

**Arbitrary Rotation Center**

- To rotate about an arbitrary point P \((px,py)\) by \(\theta\):
  - Translate the object so that P will coincide with the origin: \(T(-px,-py)\)
  - Rotate the object: \(R(\theta)\)
  - Translate the object back: \(T(px,py)\)

**Scaling Revisit**

- The standard scaling matrix will only anchor at (0,0):

\[
\begin{bmatrix}
Sx & 0 & 0 \\
0 & Sy & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

- What if I want to scale about an arbitrary pivot point?
Arbitrary Scaling Pivot

- To scale about an arbitrary pivot point P (px,py):
  - Translate the object so that P will coincide with the origin: T(-px, -py)
  - Rotate the object: S(sx, sy)
  - Translate the object back: T(px,py)

Affine Transformation

- Translation, Scaling, Rotation, Shearing are all affine transformation.
- Affine transformation – transformed point P' (x',y') is a linear combination of the original point P (x,y), i.e.
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix}
  =
  \begin{pmatrix}
  m11 & m12 & m13 \\
  m21 & m22 & m23
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  \]

- Any 2D affine transformation can be decomposed into a rotation, followed by a scaling, followed by a shearing, and followed by a translation.

Composing Transformation

- Composing Transformation – the process of applying several transformation in succession to form one overall transformation.
- If we apply transform a point P using M1 matrix first, and then transform using M2, and then M3, then we have:
  \[
  (M3 \times (M2 \times (M1 \times P))) = (M3 \times M2 \times M1 \times P)
  \]

- Matrix multiplication is associative

- Transformation products may not be commutative

- Some cases where A x B = B x A

- A x B != B x A
Transformation order matters!

- Example: rotation and translation are not commutative

  Translate (5,0) and then Rotate 60 degree
  OR
  Rotate 60 degree and then translate (5,0)??
  Rotate and then translate!!

How OpenGL does it?

- OpenGL's transformation functions are meant to be used in 3D
- No problem for 2D though - just ignore the z dimension
- Translation:
  - glVertexf(d)(tx, ty, tz) -> glVertexf(d)(tx,ty,0) for 2D

OpenGL Transformation Composition

- A global modeling transformation matrix (GL_MODELVIEW, called it M here)
  - glMatrixMode(GL_MODELVIEW)
  - The user is responsible to reset it if necessary
    - glLoadIdentity()
    - M = 1 0 0
    - 0 1 0
    - 0 0 1

How OpenGL does it?

- Rotation:
  - glVertexf(d)(angle, vx, vy, vz) ->
    - glVertexf(d)(angle, 0,0,1) for 2D

You can imagine z is pointing out of the slide.
Matrices for performing user-specified transformations are multiplied to the model view global matrix.

For example:

$$M = M \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

All the vertices $P$ defined within `glBegin()` will first go through the transformation (modeling transformation)

$$P' = M \times P$$