2D Affine Transformations

All represented as matrix operations on vectors!
Parallel lines preserved, angles/lengths not

- Scale
- Rotate
- Translate
- Reflect
- Shear

Example 1: rotation and non uniform scale on unit cube
Example 2: shear first in x, then in y

Note:
- Preserves parallels
- Does not preserve lengths and angles

2D Transforms: Translation

Rigid motion of points to new locations:

\[
\begin{align*}
    x' &= x + d_x \\
    y' &= y + d_y
\end{align*}
\]

-- Defined with column vectors:

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}
\]

\[ P' = P + T \]

2D Transforms: Scale

Stretching of points along axes:

\[
\begin{align*}
    x' &= s_x \cdot x \\
    y' &= s_y \cdot y
\end{align*}
\]

In matrix form:

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

or just: \( P' = S \cdot P \)
2D Transforms: Rotation

- Rotation of points about the origin
  \[ x' = x \cdot \cos \theta - y \cdot \sin \theta \]
  \[ y' = x \cdot \sin \theta + y \cdot \cos \theta \]

- Positive Angle: CCW
- Negative Angle: CW

Matrix form:

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

or just: \( P' = R \cdot P \)

2D Transforms: Rotation

- Substitute the 1st two equations into the 2nd two to get the general equation
  \[ x = r \cdot \cos \phi \]
  \[ y = r \cdot \sin \phi \]

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

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

Homogeneous Coordinates

- Observe: translation is treated differently from scaling and rotation

- Homogeneous coordinates: allows all transformations to be treated as matrix multiplications

Example: A 2D point \((x, y)\) is the line \((x, y, w)\), where \(w\) is any real #, in 3D homogeneous coordinates.

To get the point, homogenize by dividing by \(w\) (i.e. \(w = 1\))

Recall our Affine Transformations

- Rotation
- Translation
- Uniform Scaling
- Nonuniform Scaling
- Reflection
- Shearing

Matrix Representation of 2D Affine Transformations

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

- Scale:
  \[
  \begin{bmatrix}
    x' \\
    y'
  \end{bmatrix}
  = \begin{bmatrix}
    s_x & 0 & 0 \\
    0 & s_y & 0
  \end{bmatrix}
  \begin{bmatrix}
    x \\
    y
  \end{bmatrix}
  \]

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

- Shear:
  \[ s_{xy} = \begin{bmatrix}
    1 & 0 & 0 \\
    0 & 1 & 0
  \end{bmatrix} \]
  Reflection: \( F_y = \begin{bmatrix}
    1 & 0 & 0 \\
    0 & 0 & -1 \\
    0 & 1 & 0
  \end{bmatrix} \]

Composition of 2D Transforms

- Rotate about a point \( P_1 \)
  - Translate \( P_1 \) to origin
  - Rotate
  - Translate back to \( P_1 \)

\[
T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1)
\]

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

Original house
After translation of \( P_1 \) to origin
After rotation
After translation back to \( P_1 \)
Composition of 2D Transforms

• Scale object around point $P_1$
  – $P_1$ to origin
  – Scale
  – Translate back to $P_1$
  – Compose into $T$

$$T(x_1, y_1) \cdot S(S_x, S_y) \cdot T(-x_1, -y_1)$$

$$P' = T \cdot P$$

Composition of 2D Transforms

• Scale + rotate object around point $P_1$ and move to $P_2$
  – $P_1$ to origin
  – Scale
  – Rotate
  – Translate to $P_2$

$$P' = T \cdot P$$

The Window-to-Viewport Transformation

- Problem: Screen windows cannot display the whole world (window management)
- How to transform and clip:
  Objects to Windows to Screen

Three steps:
1. Translate
2. Scale
3. Translate

Overall Transformation:

$$M_{wV} = T(x_{\text{max}}, y_{\text{min}}) \cdot S(S_{\text{max}} - S_{\text{min}}, S_{\text{max}} - S_{\text{min}}) \cdot T(-x_{\text{max}}, -y_{\text{min}})$$

$$P' = M_{wV} \cdot P$$
3D Transformations

Representation of 3D Transformations

- Z axis represents depth
- Right Handed System
  - When looking “down” at the origin, positive rotation is CCW
- Left Handed System
  - When looking “down”, positive rotation is in CW
  - More natural interpretation for displays, big z means “far”

3D Homogenous Coordinates

- Homogenous coordinates for 2D space requires 3D vectors & matrices
- Homogenous coordinates for 3D space requires 4D vectors & matrices
- \([x,y,z,w]\)

3D Transformations: Scale

- Scale
  - Parameters for each axis direction
  
  \[
  S(s_x, s_y, s_z) = \begin{bmatrix}
  s_x & 0 & 0 & 0 \\
  0 & s_y & 0 & 0 \\
  0 & 0 & s_z & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

3D Transformations: Translate

- Translation
  
  \[
  T(d_x, d_y, d_z) = \begin{bmatrix}
  1 & 0 & 0 & d_x \\
  0 & 1 & 0 & d_y \\
  0 & 0 & 1 & d_z \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

3D Transformations: Rotation

- One rotation for each world coordinate axis
  
  \[
  P' = R \cdot P
  \]

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

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

  \[
  R_z(\theta) = \begin{bmatrix}
  \cos \theta & -\sin \theta & 0 & 0 \\
  \sin \theta & \cos \theta & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]
Rotation Around an Arbitrary Axis

- Rotate a point P around axis \( \mathbf{n} (x,y,z) \) by angle \( \theta \)
- \( \mathbf{R} = \begin{bmatrix} x^2 + c & xy + sc & xz - sy \\ yx - sc & y^2 + c & yz + sx \\ zx + sy & zy - sx & z^2 + c \end{bmatrix} \)
  \[ c = \cos(\theta) \]
  \[ s = \sin(\theta) \]
  \[ t = (1 - c) \]

Improved Rotations

- Euler Angles have problems
  - How to interpolate keyframes?
  - Angles aren’t independent
  - Interpolation can create Gimble Lock, i.e. loss of a degree of freedom when axes align
- Solution: Quaternions!

Quaternions

- Matrices are not the only (or best) way of representing rotations. For one thing, they are redundant (9 numbers instead of 3) and, for another, they are difficult to manipulate.
- Quaternions are a compact way of representing rotations. They are a 4-dimensional, real vector that can be used to rotate a vector in a 3-dimensional space.

Rotation by Quaternion

- \( \mathbf{q}_0 \mathbf{p} \mathbf{q}^{-1} \) where \( |q| = 1 \)
- \( \mathbf{R}_0 \mathbf{p} = \begin{bmatrix} 0 & -(\mathbf{p} \cdot \mathbf{n}) & \mathbf{p} \times \mathbf{n} \\ -\mathbf{p} \times \mathbf{n} & \mathbf{I} - 2(\mathbf{n} \times \mathbf{n}) \\ \mathbf{n} \times \mathbf{p} & 2(\mathbf{n} \times \mathbf{p}) \\ \end{bmatrix} \)

Recognize this? It is the Rodrigues formula!
Quaternion Composition
Since a quaternion basically stores the axis vector and angle of rotation, it is not surprising that we can write the components of a rotation matrix using the quaternion components.

\[ q = (\cos(\theta/2), \sin(\theta/2)) = (x, y, z, w) \]

\[
R_q = \begin{pmatrix}
1 - 2y^2 - 2z^2 & 2xy - 2zw & 2xz + 2yw & 0 \\
2xy + 2zw & 1 - 2x^2 - 2z^2 & 2yz - 2wx & 0 \\
2xz - 2yw & 2yz + 2wx & 1 - 2x^2 - 2y^2 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Crucially, the composition of two rotations given by quaternions is simply their quaternion product.

\[ R_q(R_p(q)) = R_p(q') \quad \text{where } q' = q\bar{q} \]

- Note that the product of two unit quaternions is another unit quaternion.
- Note that quaternion multiplication, like matrix multiplication, is not commutative.

From rotation matrix to quaternion
Given \( R = (r_{ij}) \), solve expression on previous page for quaternion elements \( q \).
Linear combinations of diagonal elements seem to solve the problem:

\[
q_x = \frac{1}{3}(1 + r_{11} + r_{22} + r_{33}) \\
q_y = \frac{1}{3}(1 + r_{11} - r_{22} - r_{33}) \\
q_z = \frac{1}{3}(1 - r_{11} + r_{22} - r_{33}) \\
q_w = \frac{1}{3}(1 - r_{11} - r_{22} + r_{33})
\]

so take four square roots and you’re done? You have to figure the signs out. There is a better way …

Quaternion Interpolation
One of the main motivations for using quaternions in graphics is the case with which we can define interpolation between two orientations. Think, for example, about moving a camera smoothly between two views.

A & B are quaternions

\[
C(t) = \text{slerp}(A, B, t) = A\sin(\Omega t) + B\sin(\Omega t) \sin(\Omega t)
\]

slerp – Spherical linear interpolation
Need to take equals steps on the sphere

What about interpolating multiple keyframes?
- Shoemake suggests using Bezier curves on the sphere
- Offers a variation of the De Casteljau algorithm using slerp and quaternion control points

3D Transformations: Reflect

- Reflection:
  \[
  F_x = \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & -1 & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]
  about x-y plane

- Reflection:
  \[
  F_y = \begin{pmatrix}
  -1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]
  about y-z plane

Reflection

corresponds to negative scale factors

\[
s_x = -1 \quad s_y = -1 \\
s_x = -1 \quad s_y = -1
\]

3D Transformations: Shear

- Shear: (function of y)
  \[
  H = \begin{pmatrix}
  1 & sh_x & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1 
  \end{pmatrix}
  \]
  \[x' = x + sh_x \cdot y\]

Example: Composition of 3D Transformations

- Goal: Transform \( P_1 P_2 \) and \( P_1 P_3 \)

  (a) Initial position  
  (b) Final position

Final Result

- What we’ve really done is transform the local coordinate system \( R_x, R_y, R_z \) to align with the origin \( x, y, z \)

Example 2: Composition of 3D Transformations

- Airplane defined in \( x,y,z \)
- Problem: want to point it in Dir of Flight (DOF) centered at point \( P \)
- Note: DOF is a vector
- Process:
  - Rotate plane
  - Move to \( P \)
Example 2 (cont.)

- $Z_p$ axis to be DOF
- $X_p$ axis to be a horizontal vector perpendicular to DOF
- $Y_p$, vector perpendicular to both $Z_p$ and $X_p$ (i.e. $Z_p \times X_p$)

Transformations to Change Coordinate Systems

- Issue: the world has many different relative frames of reference
- How do we transform among them?
- Example: CAD Assemblies & Animation Models

Transformations to Change Coordinate Systems

- 4 coordinate systems
- 1 point $P$
- $M_{1\to 2} = T(4,2)$
- $M_{2\to 3} = T(2,3) \cdot S(0.5,0,0.5)$
- $M_{3\to 4} = T(6.7,1.8) \cdot R(45^\circ)$

$$M_{i\to k} = M_{i\to j} \cdot M_{j\to k}$$

Coordinate System Example (1)

- Translate the House to the origin
- $M_{5\to 2} = T(x_1, y_1)$
- $M_{5\to 1} = (M_{5\to 2})^{-1} = T(-x_1, -y_1)$

The matrix $M_{i}$ that maps points from coordinate system $j$ to $i$ is the inverse of the matrix $M_{j}$ that maps points from coordinate system $j$ to coordinate system $i$.

Coordinate System Example (2)

- Transformation Composition:
- $M_{5\to 1} = M_{5\to 4} \cdot M_{4\to 3} \cdot M_{3\to 2} \cdot M_{2\to 1}$

World Coordinates and Local Coordinates

- To move the tricycle, we need to know how all of its parts relate to the WCS
- Example: front wheel rotates on the ground wrt the front wheel’s z axis:
  Coordinates of $P$ in wheel coordinate system:
  $$P^{(wh)} = R(\alpha) \cdot P^{(wh)}$$
Properties of Transformation Matrices

• Note that matrix multiplication is not commutative
• i.e. in general $M_1M_2 \neq M_2M_1$

• $T$ – reflection around y axis
• $T'$ – rotation in the plane