3D Coordinate Systems

- 3D computer graphics involves the additional dimension of depth, allowing more realistic representations of 3D objects in the real world.
- There are two possible ways of "attaching" the Z-axis, which gives rise to a left-handed or a right-handed system.

3D Transformation

- The translation, scaling and rotation transformations used for 2D can be extended to three dimensions.
- In 3D, each transformation is represented by a 4x4 matrix.
- Using homogeneous coordinates it is possible to represent each type of transformation in a matrix form and integrate transformations into one matrix.
- To apply transformations, simply multiply matrices, also easier in hardware and software implementation.
- Homogeneous coordinates can represent directions.
- Homogeneous coordinates also allow for non-affine transformations, e.g., perspective projection.

Homogeneous Coordinates

- In 2D, use three numbers to represent a point.
- \((x,y) = (wx,wy,w)\) for any constant \(w \neq 0\).
- To go backwards, divide by \(w\), \((x,y)\) becomes \((x,y,1)\).
- Transformation can now be done with matrix multiplication.

\[
\begin{bmatrix}
    x' \\
    y' \\
    1
\end{bmatrix}
= \begin{bmatrix}
    a_x & a_y & b_x & x \\
    a_x & a_y & b_y & y \\
    0 & 0 & 1 & 1
\end{bmatrix}
\]

Basic 2D Transformations

- Translation:
  \[
  \begin{bmatrix}
    1 & 0 & b_x \\
    0 & 1 & b_y \\
    0 & 0 & 1
  \end{bmatrix}
  \]

- Scaling:
  \[
  \begin{bmatrix}
    s_x & 0 & 0 \\
    0 & s_y & 0 \\
    0 & 0 & 1
  \end{bmatrix}
  \]

- Rotation:
  \[
  \begin{bmatrix}
    \cos\theta & -\sin\theta & 0 \\
    \sin\theta & \cos\theta & 0 \\
    0 & 0 & 1
  \end{bmatrix}
  \]
Translation and Scaling Matrices

- The translation and scaling transformations may be represented in 3D as follows:

  **Translation matrix**
  \[
  \begin{bmatrix}
  1 & 0 & 0 & \text{tr}_x \\
  0 & 1 & 0 & \text{tr}_y \\
  0 & 0 & 1 & \text{tr}_z \\
  0 & 0 & 0 & 1 \\
  \end{bmatrix}
  \]

  **Scaling matrix**
  \[
  \begin{bmatrix}
  S_x & 0 & 0 & 0 \\
  0 & S_y & 0 & 0 \\
  0 & 0 & S_z & 0 \\
  0 & 0 & 0 & 1 \\
  \end{bmatrix}
  \]

**Translation**

A translation vector \( \mathbf{V} \) is defined by its components as \( \mathbf{V} = ai + bj + ck \)

In homogeneous matrix form this is

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  1 \\
  \end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 & a \\
  0 & 1 & 0 & b \\
  0 & 0 & 1 & c \\
  0 & 0 & 0 & 1 \\
  \end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1 \\
  \end{bmatrix}
\]

\( \mathbf{V} = (ai + bj + ck) = (a, b, c) \)

**Scaling**

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  1 \\
  \end{bmatrix}
=\begin{bmatrix}
  a & 0 & 0 & 0 \\
  0 & b & 0 & 0 \\
  0 & 0 & c & 0 \\
  0 & 0 & 0 & 1 \\
  \end{bmatrix}\begin{bmatrix}
  x \\
  y \\
  z \\
  1 \\
  \end{bmatrix}
\]

**3D Shearing**

Shearing:

The change in each coordinate is a linear combination of all three

Transforms a cube into a general parallelepiped

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  1 \\
  \end{bmatrix} =
\begin{bmatrix}
  1 & a & b & 0 \\
  c & 1 & d & 0 \\
  e & f & 1 & 0 \\
  0 & 0 & 0 & 1 \\
  \end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1 \\
  \end{bmatrix}
\]
Rotation

- In 2D, rotation is about a point.
- In 3D, rotation is about a vector, which can be done through rotations about x, y or z axes.
- Positive rotations are anti-clockwise, negative rotations are clockwise, when looking down a positive axis towards the origin.

Major Axis Rotation Matrices

- about X axis
  \[
  R_x = \begin{bmatrix}
  1 & 0 & 0 \\
  0 & \cos \theta & -\sin \theta \\
  0 & \sin \theta & \cos \theta 
  \end{bmatrix}
  \]
  Rotations are orthogonal matrices, preserving distances and angles.

- about Y axis
  \[
  R_y = \begin{bmatrix}
  \cos \theta & 0 & \sin \theta \\
  0 & 1 & 0 \\
  -\sin \theta & 0 & \cos \theta 
  \end{bmatrix}
  \]

- about Z axis
  \[
  R_z = \begin{bmatrix}
  \cos \theta & -\sin \theta & 0 \\
  \sin \theta & \cos \theta & 0 \\
  0 & 0 & 1 
  \end{bmatrix}
  \]

Rotation Axis

- In general rotation vector does not pass through origin.

Translate P1 to force the axis to pass through the origin.
Rotation about an Arbitrary Axis

Basic Idea
1. Translate \((x_1, y_1, z_1)\) to the origin
2. Rotate \((x'_2, y'_2, z'_2)\) onto the z axis
3. Rotate the object around the z-axis
4. Rotate the axis to the original orientation
5. Translate the rotation axis to the original position

\[
\begin{align*}
[T_{Rarb}] &= [T_{R_{z'}}]^{-1}[T_{R_{z}}]^{-1} [T_{R_{z}}(\phi)] [T_{R_{z}}(\alpha)] [T_{R_{z}}(\theta)] [T_{R_{z}}] [T_{R_{z}}]^{-1}
\end{align*}
\]
Rotation about an Arbitrary Axis

- Step 4. Rotate about \( z \) axis by the desired angle \( \theta \)

\[
[T_{z,\alpha}] = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

- Step 5. Apply the reverse transformation to place the axis back in its initial position

\[
[T_{x,\alpha}^{-1}]^{-1}[T_{y,\alpha}^{-1}]^{-1}[T_{z,\alpha}^{-1}]^{-1}[T_{y,\alpha}][T_{x,\alpha}][T_{z,\alpha}][T_{x,\alpha}]
\]

---

Rotation about an Arbitrary Axis

Find the new coordinates of a unit cube 90°-rotated about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

- Step 1. Translate point A (2,1,0) to the origin

\[
[T_{T}] = \begin{bmatrix}
1 & 0 & 0 & -2 \\
0 & 1 & 0 & -1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

---
Rotation about an Arbitrary Axis

0 Step 2. Rotate axis A'B' about the x axis by an angle \( \alpha \), until it lies on the \( xz \) plane.

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \alpha & -\sin \alpha \\
0 & \sin \alpha & \cos \alpha
\end{bmatrix}
\]

Projected point \((0,1,2,1)\)

\[\mathbf{R}(1,2,1)\]

Rotation about an Arbitrary Axis

0 Step 3. Rotate axis A'B'' about the y axis by an angle \( \phi \), until it coincides with the z axis.

\[
\begin{bmatrix}
\cos \phi & 0 & \sin \phi \\
0 & 1 & 0 \\
-\sin \phi & 0 & \cos \phi
\end{bmatrix}
\]

Projected point \((0,0,\sqrt{5})\)

\(\mathbf{B}'(1,0,\sqrt{5})\)

Finally, the concatenated rotation matrix about the arbitrary axis AB becomes,

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \phi & 0 & -\sin \phi \\
0 & 0 & 1 & 0 \\
0 & \sin \phi & 0 & \cos \phi
\end{bmatrix}
\]

Rotation about an Arbitrary Axis

0 Step 4. Rotate the cube 90° about the z axis

Finally, the concatenated rotation matrix about the arbitrary axis AB becomes,

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]
Rotation about an Arbitrary Axis

- Multiplying $[T_{arb}]_A$ by the point matrix of the original cube

$$[P] = [T_{arb}] [P]$$

$$\begin{bmatrix}
0.166 & -0.075 & 0.993 & 1.742 \\
0.742 & 0.667 & 0.075 & -1.153 \\
0.065 & 0.416 & 0.917 & 0.960 \\
0 & 0 & 0 & 1
\end{bmatrix}$$

- Reflection Relative to the xy Plane

- Z-axis Shear

Q1 - Translate by <1, 1, 1>

- A translation by an offset (tx, ty, tz) is achieved using the following matrix:

$$M_r (t_x, t_y, t_z) = \begin{bmatrix}
1 & 0 & 0 & t_x \\
0 & 1 & 0 & t_y \\
0 & 0 & 1 & t_z \\
0 & 0 & 0 & 1
\end{bmatrix}$$

- So to translate by a vector (1, 1, 1), the matrix is simply:

$$M_r (1, 1, 1) = \begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1
\end{bmatrix}$$

Q2 - Rotate by 45 degrees about x axis

- So to rotate by 45 degrees about the x-axis, we use the following matrix:

$$R_x (45) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \sqrt{2}/2 & -\sqrt{2}/2 & 0 \\
0 & \sqrt{2}/2 & \sqrt{2}/2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}$$
Q3 - Rotate by 45 about axis <1, 1, 1>

- So a rotation by 45 degrees about <1, 1, 1> can be achieved by a few successive rotations about the major axes. Which can be represented as a single composite transformation.

\[
d = \sqrt{n_{z}^2 + n_{y}^2} = \sqrt{2} = 1.414
\]

\[
n_{x} = 1
\]

\[
n_{y} = 1 \text{ SO } \beta = \tan^{-1}\frac{n_{y}}{d} = \tan^{-1}\frac{1}{\sqrt{2}} = 35.264
\]

\[
n_{z} = 1 \text{ SO } \alpha = \tan^{-1}\frac{n_{z}}{n_{x}} = \tan^{-1}\frac{1}{1} = 45
\]

Q3 - Arbitrary Axis Rotation

- The composite transformation can then be obtained as follows:

\[
M_{x}(1,1,1) = R^{-1}_{x}(\alpha) \bullet R^{-1}_{y}(\beta) \bullet R_{z}(\phi) \bullet R_{x}(\alpha)
\]

\[
= \begin{bmatrix}
\cos(45) & 0 & \sin(45) & 0 \\
0 & 1 & 0 & 0 \\
-\sin(45) & 0 & \cos(45) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Directions vs. Points

- We have looked at transforming points.
- Directions are also important in graphics.
- Viewing directions.
- Normal vectors.
- Ray directions.

- Directions are represented by vectors, like points, and can be transformed, but not like points.
- Say we define a direction as the difference of two points: \( d = a - b \). This represents the direction of the line between two points.
- Now we translate the points by the same amount: \( a' = a + t, b' = b + t \).
- Have we transformed \( d \)?

Homogeneous Directions

- Translation does not affect directions!
- Homogeneous coordinates give us a clear way of handling this, e.g., direction \((x,y)\) becomes homogeneous direction \((x,y,0)\), and remains the same after translation:

\[
\begin{bmatrix}
1 & 0 & b \\
0 & 1 & b \\
0 & 0 & 1
\end{bmatrix}
\]

- \((x, y, 0)\) is a vector, \((x,y,1)\) is a point.
- The same applies to rotation and scaling, e.g., scaling changes the length of vector, but not direction.
- Normal vectors are slightly different though (can’t always use the matrix for points to transform the normal vector).
Alternative Rotations

- Specify the rotation axis and the angle (OpenGL method)
- Euler angles: Specify how much to rotate about X, then how much about Y, then how much about Z.
- These are hard to think about, and hard to compose.
- Quaternions:
  - 4-vector related to axis and angle, unit magnitude, e.g., rotation about axis (nx, ny, nz) by angle θ:
    $\{n_x \cos(\theta/2), n_y \cos(\theta/2), n_z \cos(\theta/2), \sin(\theta/2)\}$
  - Only normalized quaternions represent rotations, but you can normalize them just like vectors, so it isn't a problem.
  - But we don't want to learn all the maths about quaternions in this module, because we have to learn how to create a basic application before trying to make rotation faster.

OpenGL Transformations

- OpenGL internally stores two matrices that control viewing of the scene:
  - The GL_MODELVIEW matrix for modeling and world to view transformations.
  - The GL_PROJECTION matrix captures the view to canonical conversion.
  - Mapping from canonical view volume into window space is through a glViewport function call.
- Matrix calls, such as glRotate, glTranslate, glScale right multiply the transformation matrix M with the current matrix C (e.g., identity matrix initially), resulting in CM - the last one is the first applied.