COMP30019 Graphics and Interaction Three-dimensional transformation geometry and perspective

Adrian Pearce

Department of Computing and Information Systems
University of Melbourne

The University of Melbourne

Lecture outline

Three-dimensional transformations and perspective

How do three-dimensional transformations differ from two-dimensional transformations?

Aim: understand how to transform arbitrary perspectives in three-dimensions (3D).

Reading:

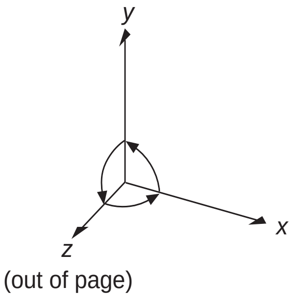
- Foley Sections 5.7 matrix representation of 3D transformations and 5.8 composition of 3D transformations.

Three-dimensional coordinate systems

3D (Cartesian) coordinates (X, Y, Z) can be considered in terms of left-handed and right-handed coordinate systems. The “right-hand rule”:

thumb	X axis
index finger	Y axis
big finger	Z axis

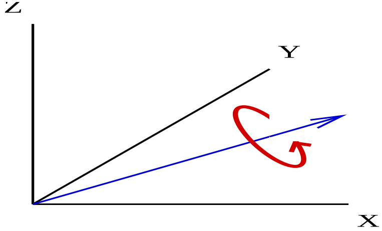
Rotation about axis in the right-handed coordinate system



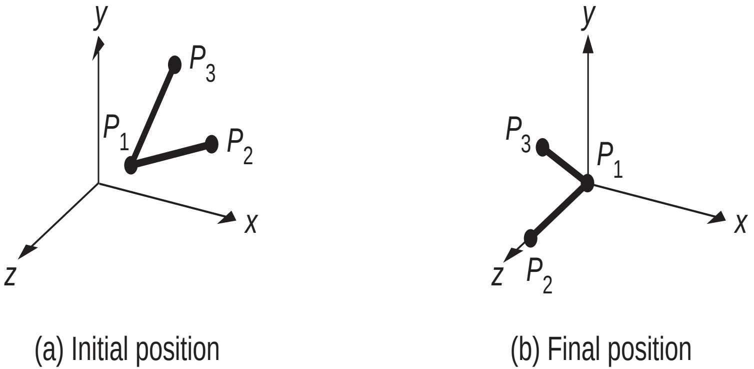
Axis of rotation	Direction of positive rotatior
x	y to z
y	z to x
z	x to y

Three-dimensional rotation

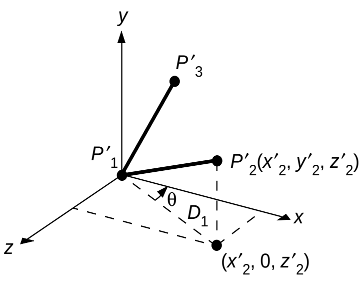
Need to specify axis of rotation, and by how much such as a 3D unit vector, plus rotation angle.
Convention: rotation angle is clockwise looking along axis of rotation
Note, this is consistent with ordinary 2D anti-clockwise convention:
Really a rotation around a positive *Z* axis coming up out of the page in a right-handed coordinate frame.

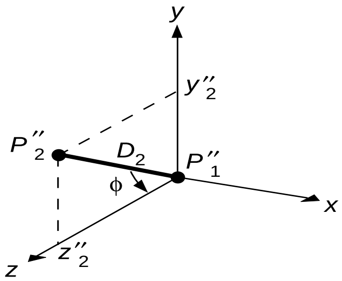


Example: rotation of a simple object

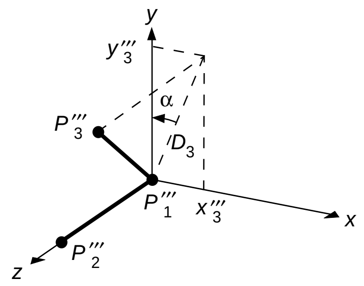


Example from Section 5.8 of Foley.





Rotation about the *y* axis (left) by $-(90-\theta)$ followed by rotation around the *x* axis (right) by ϕ (shown for P_1P_3 only).



Final rotation about the *z* axis is by α.
The composite matrix *R* for the overall transformation is calculated by multiplying the individual rotation matrices and translation matrix

$$R_z(\alpha) \cdot R_x(\phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) = R \cdot T$$

Rotations about an arbitrary axis

Two rotations about coordinate axes to line up axis of rotation with one of the coordinate axes.

One rotation around this last axis, two rotations to undo the first two, and put the axis of rotation back where it was.

If axis doesn't pass through the origin, you'll need to wrap an appropriate translation and its inverse around all this.

Homogeneous versions

In 3D the point in homogeneous coordinates

$$(X, Y, Z, W)$$

corresponds to the Cartesian point
$(X/W, Y/W, Z/W)$ for $W \neq 0$

Completely analogous to the 2D case, where $W = 1$.
Translation:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Scaling: $\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x_s & 0 & 0 & 0 \\ 0 & y_s & 0 & 0 \\ 0 & 0 & z_s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$

Rotation counterclockwise using right-handed coordinate system about *Z* axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation around the *y* axis:

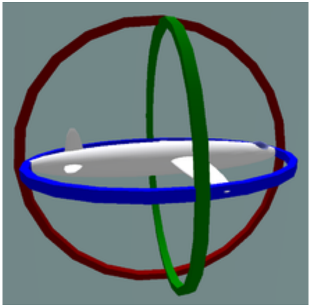
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

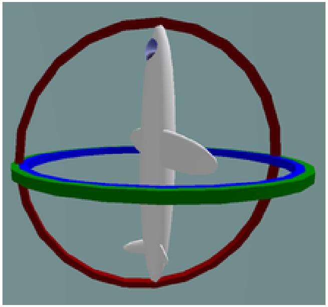
Rotation around the *x* axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Note cyclic wrap-around of factors for each axis.

Gimbal Lock





Gimbal Lock

- If we combine rotation matrices as $R = R_x(\phi) \cdot R_y(\theta) \cdot R_z(\alpha)$ we have

$$\begin{bmatrix} cos(\theta)cos(\alpha) & cos(\theta)sin(\gamma) & sin(\theta) & 0 \\ sin(\phi)sin(\theta)cos(\alpha) + cos(\phi)sin(\alpha) & -sin(\phi)sin(\theta)sin(\alpha) + cos(\phi)cos(\alpha) & -sin(\phi)cos(\theta) & 0 \\ -cos(\phi)sin(\theta)cos(\alpha) + sin(\phi)sin(\alpha) & cos(\phi)sin(\theta)sin(\alpha) + sin(\phi)cos(\alpha) & cos(\theta)cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Now consider $\theta = \frac{\pi}{2}$:

$$R = \begin{bmatrix} 0 & 0 & 1 & 0 \\ \sin\alpha\cos\gamma + \cos\alpha\sin\gamma & -\sin\alpha\sin\gamma + \cos\alpha\cos\gamma & 0 & 0 \\ -\cos\alpha\cos\gamma + \sin\alpha\sin\gamma & \cos\alpha\sin\gamma + \sin\alpha\cos\gamma & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 0 & 0 & 1 & 0 \\ \sin(\alpha+\gamma) & \cos(\alpha+\gamma) & 0 & 0 \\ -\cos(\alpha+\gamma) & \sin(\alpha+\gamma) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Lost a degree of freedom.
- Changing the rotation order affects the axis on which gimbal lock occurs, but does not prevent it from occurring

Quaternions

- One way to avoid this problem is by using Quaternions to calculate rotations rather than Euler angles.
- Quaternions represent an extension on complex numbers of the form $q = s + v_1\boldsymbol{i} + v_2\boldsymbol{j} + v_3\boldsymbol{k}$ where $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ are imaginary orthogonal unit vectors. This is often written as $q = (s, \boldsymbol{v})$
- Quaternion rules:
  - $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1$
  - $\boldsymbol{ij = k, ji = -k}$
  - $\boldsymbol{jk = i, kj = -i}$
  - $\boldsymbol{ki = j, ik = -j}$
  - $|q| = \sqrt{s^2 + \boldsymbol{v}^2}$

Quaternion Operations

Quaternion multiplication works as follows:

$$\begin{aligned} q_1 q_2 &= (s_1 + v_{11}\boldsymbol{i} + v_{12}\boldsymbol{j} + v_{13}\boldsymbol{k})(s_2 + v_{21}\boldsymbol{i} + v_{22}\boldsymbol{j} + v_{23}\boldsymbol{k}) \\ &= s_1 s_2 + s_1 v_{21}\boldsymbol{i} + s_2 v_{22}\boldsymbol{j} + s_3 v_{23}\boldsymbol{k} + \ldots + v_{13} v_{22}\boldsymbol{kj} + v_{13} v_{23}\boldsymbol{k}^2 \end{aligned}$$

When we collect terms and apply the rules from the previous page, we get:

$$q_1 q_2 = (s_1 s_2 - \boldsymbol{v_1} \cdot \boldsymbol{v_2}, s_1 \boldsymbol{v_2} + s_2 \boldsymbol{v_1} + \boldsymbol{v_1} \times \boldsymbol{v_2})$$

Quaternion multiplication is *not* commutative

The *conjucate* of a quaternion $q = (s, \boldsymbol{v})$ is $q* = (s, -\boldsymbol{v})$
The *inverse* of q is $q^{-1} = \frac{q*}{|q|^2}$

Quaternion Rotations

To rotate a point (x, y, z) by angle θ counterclockwise around an arbitrary axis defined by unit vector u we compute:

$$P' = qPq^{-1}$$

where $P = (0, x, y, z)$ is a quaternion that represents the point
$q = (cos(\frac{\theta}{2}), sin(\frac{\theta}{2})\boldsymbol{u})$
$q^{-1} = (cos(\frac{\theta}{2}), -sin(\frac{\theta}{2})\boldsymbol{u})$

We can compute multiple rotations as
$P' = q_n \dots q_2 q_1 P {q_1}^{-1} {q_2}^{-1} \dots {q_n}^{-1}$

Quaternions to matrices

We can represent the quaternion as a matrix:

$$R = \begin{bmatrix} s^2 + {v_1}^2 - {v_2}^2 - {v_3}^2 & 2v_1 v_2 - 2sv_3 & 2sv_2 + 2v_1 v_3 & 0 \\ 2sv_3 + 2v_1 v_2 & s^2 - {v_1}^2 + {v_2}^2 - {v_3}^2 & 2v_2 v_3 - 2sv_1 & 0 \\ 2v_1 v_3 - 2sv_2 & 2sv_1 + 2v_2 v_3 & s^2 - {v_1}^2 - {v_2}^2 + {v_3}^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(for unit quaternions)

Activity

Use quaternions to rotate the point (-1, 2, 1) by 90 degrees around the X axis.

Why Quaternions?

Quaternions have a number of advantages over rotation matrices

- Avoids gimbal lock
- More efficient to calculate
- Not dependant on axis rotation order
- Easy to interpolate rotations

Arbitrary camera orientation

Suppose objects are in some world coordinate system, consider the pose (position and orientation) of a camera in this system.

An objects' position must be transformed into camera-centred coordinates before a projection can be done, requiring

- a translation (to put cameras optical centre at origin)
- a three-dimensional rotation to line up Z-axis with optical axis (requires two coordinate axis rotations)
- *another* rotation around Z-axis to line up X and Y axes (which way is *up*)

Matrix form of perspective projection

Perspective projection can be expressed as a matrix operation in homogeneous coordinates, where the 3D point (X, Y, Z) is represented by the homogeneous coordinates (wX, wY, wZ, w), where $w \neq 0$ and is typically 1.

Hint: think of perspective projection as a mapping from 3D to 2D, then it can be expressed as a multiplication by a 4×3 matrix (that is, 3 rows, 4 columns).

We want the homogeneous point $(X, Y, Z, 1)$ in 3D to "go to" the homogeneous point $(fX/Z, fY/Z, 1)$ in 2D—this comes straight from the equations of perspective projection.

This latter point is the same as $(X, Y, Z/f)$, since multiplication throughout by the constant Z/f gives the same point in homogeneous coordinates, and

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z/f \end{bmatrix} \to \begin{bmatrix} fX/Z \\ fY/Z \\ 1 \end{bmatrix}$$

- Rightarrow is just homogenisation (dividing through to make homogeneous coordinate equal to one (1)!

Perspective projection can also be thought of as a mapping from one 3D vector to another 3D vector, given by a 4×4 transformation matrix using a monotonic function of depth, called *pseudo-depth*.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ 1 \\ Z/f \end{bmatrix} \rightarrow \begin{bmatrix} fX/Z \\ fY/Z \\ f/Z \\ 1 \end{bmatrix}$$

Pseudo-depth can make perspective projection into an invertible affine transformation (as it stores *unused Z* dimension in two-dimensions until you need it to go back to three-dimensions again).

- Pseudo-depth can be used conveniently for ordering points in hidden-surface elimination, and similar processes.

Summary

- In three-dimensional transformations the convention is to rotate clockwise looking along axis of rotation, using the *right-hand* rule for coordinates.
- Rotations about an arbitrary axis involves two rotations to line up axis of rotation with a coordinate axis, a rotation around this last axis then two rotations to undo the first two (and some translations if rotation axis does not pass through origin).
- Projections from three-dimensions to two-dimensions can be conveniently represented as matrix transformations, allowing for projection variants according to matrix factors.