Week 8:
3D Computational Geometry II

COMP270: Mathematics for 3D Worlds & Simulations

BSc(Hons) Computing for Games

“Unfortunately, no-one can be told what the matrix is. You have to see it for yourself”

- *Morpheus*

Matrices in 3D

Matrix multiplication

Multiplying two 3x3 matrices:

$$\begin{pmatrix} \boxed{a_{11} \quad a_{12} \quad a_{13}} \\ a_{21} \quad a_{22} \quad a_{23} \\ \boxed{a_{31} \quad a_{32} \quad a_{33}} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

$$= \begin{pmatrix} \boxed{a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} & a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} & \boxed{a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}} & a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{pmatrix}$$

Matrix multiplication

Multiplying two 3x3 matrices:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} & a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} & a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{pmatrix}$$

Matrix multiplication

Multiplying a 3x3 matrix with a 3x1 vector:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y + a_{13}z \\ a_{21}x + a_{22}y + a_{23}z \\ a_{31}x + a_{32}y + a_{33}z \end{pmatrix}$$

Matrix multiplication

Multiplying a 3x3 matrix with a 3x1 vector:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y + a_{13}z \\ a_{21}x + a_{22}y + a_{23}z \\ a_{31}x + a_{32}y + a_{33}z \end{pmatrix}$$

Inverse of a 3x3 matrix

Method of cofactors:

1. Calculate the *matrix of minors*
2. Convert it to the *matrix of cofactors*
3. Find the *adjugate*
4. Divide by the *determinant*…

http://wwwf.imperial.ac.uk/metric/metric_public/matrices/inverses/inverses2.html

https://www.khanacademy.org/math/algebra-home/alg-matrices#alg-determinants-and-inverses-of-large-matrices

"3D Math Primer for Graphics and Game Development", Chapter 6

3D homogeneous matrices

- A 3x3 matrix can represent a mapping that is a *linear combination* of the three coordinate values (x, y, and z) only
  - i.e. the matrix $\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$ applied to the vector $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ represents a function $f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$ where:

$$x' = m_{11}x + m_{12}y + m_{13}z$$
$$y' = m_{21}x + m_{22}y + m_{23}z$$
$$z' = m_{31}x + m_{32}y + m_{33}z$$

- What if we want to include a constant value (in geometrical terms, a translation)?
  - An *affine* transformation

3D homogeneous matrices

Applying a 4x4 homogeneous matrix to a point/vector:

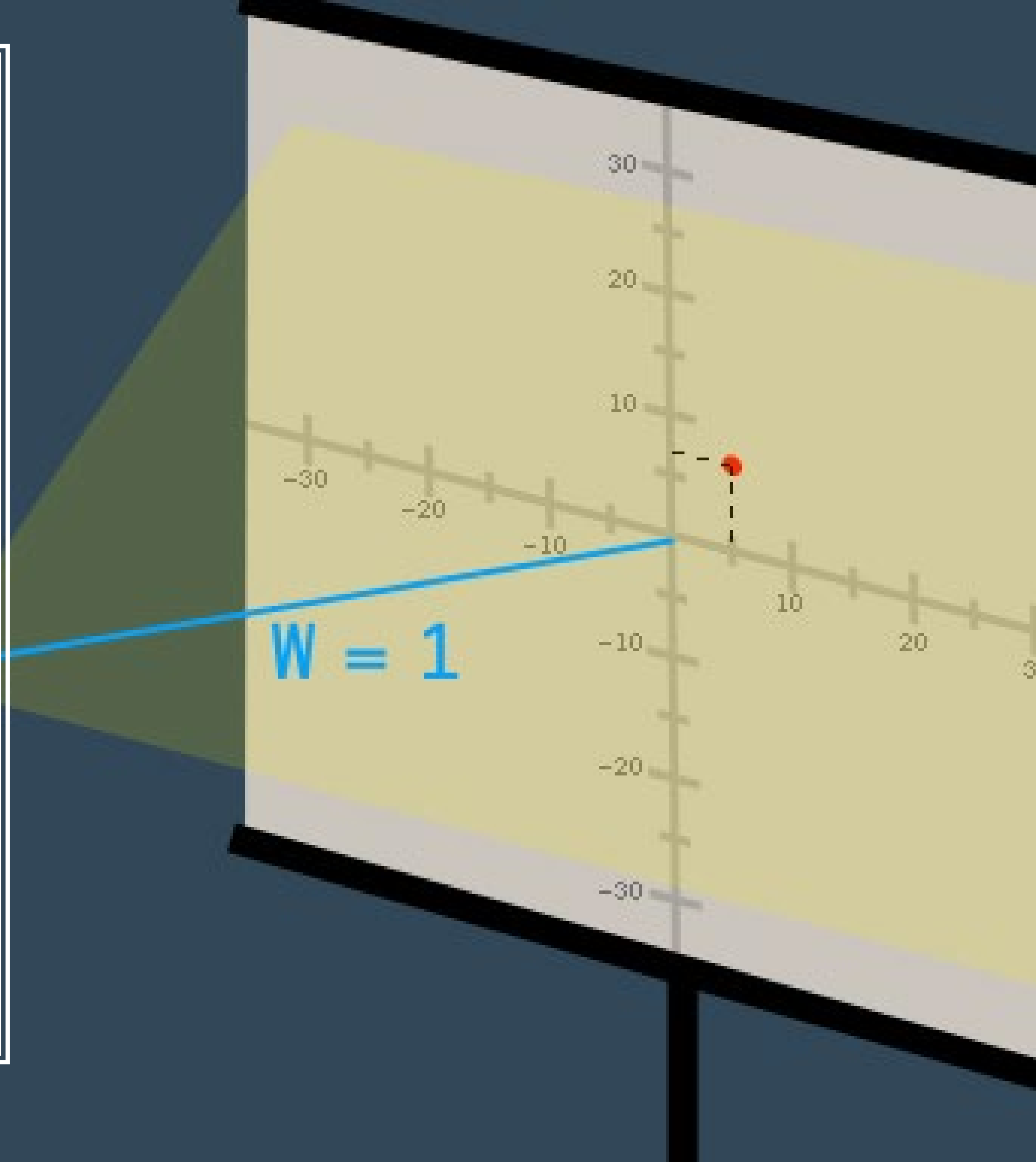
$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} r_{11}x + r_{12}y + r_{13}z + \boxed{t_x w} \\ r_{21}x + r_{22}y + r_{23}z + \boxed{t_y w} \\ r_{31}x + r_{32}y + r_{33}z + \boxed{t_z w} \\ w \end{pmatrix}$$

Note that only the w coordinate is affected by the 4th column (translation values)

- For points (which have a position), $w = 1$
- For vectors (which have only direction), $w = 0$

What is w?

- An “extra dimension” (not time!) added to allow translations…
- Extends 3D space to *projective space*
- A scaling factor/“distance to the projector”:
  - $(x, y, z, w) \rightarrow \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$
  - w = 1: direct mapping of a point to 3D space
  - w = 0: a point that is infinitely far away/a vector with infinite length
- More info:
  - https://www.tomdalling.com/blog/modern-opengl/explaining-homogenous-coordinates-and-projective-geometry/
  - https://hackernoon.com/programmers-guide-to-homogeneous-coordinates-73cbfd2bcc65



3D homogeneous matrices

Multiplying two 4x4 homogeneous matrices:

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} s_{11} & s_{12} & s_{13} & u_x \\ s_{21} & s_{22} & s_{23} & u_y \\ s_{31} & s_{32} & s_{33} & u_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} r_{11}s_{11} + r_{12}\, s_{21} + r_{13}\, s_{31} & r_{11}s_{12} + r_{12}\, s_{22} + r_{13}\, s_{32} & r_{11}s_{13} + r_{12}s_{23} + r_{13}\, s_{33} & r_{11}u_x + r_{12}u_y + r_{13}u_z + t_x \\ r_{21}s_{11} + r_{22}\, s_{21} + r_{23}\, s_{31} & r_{21}s_{12} + r_{22}\, s_{22} + r_{23}\, s_{32} & r_{21}s_{13} + r_{22}\, s_{23} + r_{23}\, s_{33} & r_{21}u_x + r_{22}u_y + r_{23}u_z + t_y \\ r_{31}s_{11} + r_{32}\, s_{21} + r_{33}\, s_{31} & r_{31}s_{12} + r_{32}\, s_{22} + r_{33}\, s_{32} & r_{31}s_{13} + r_{32}\, s_{23} + r_{33}\, s_{33} & r_{31}u_x + r_{32}u_y + r_{33}u_z + t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D homogeneous matrices

Applying a rotation and then a translation:

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_{11} & s_{12} & s_{13} & 0 \\ s_{21} & s_{22} & s_{23} & 0 \\ s_{31} & s_{32} & s_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} s_{11} & s_{12} & s_{13} & t_x \\ s_{21} & s_{22} & s_{23} & t_y \\ s_{31} & s_{32} & s_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D homogeneous matrices

Applying a translation and then a rotation:

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & u_x \\ 0 & 1 & 0 & u_y \\ 0 & 0 & 1 & u_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{11}u_x + r_{12}u_y + r_{13}u_z \\ r_{21} & r_{22} & r_{23} & r_{21}u_x + r_{22}u_y + r_{23}u_z \\ r_{31} & r_{32} & r_{33} & r_{31}u_x + r_{32}u_y + r_{33}u_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D homogeneous rotation matrices

In a right-handed coordinate system:

Rotation (anticlockwise) about the x-axis:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation (anticlockwise) about the y-axis:

$$R_y(\theta) = \begin{pmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation (anticlockwise) about the z-axis:

$$R_z(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse of a transformation matrix

- Rule 1: the inverse of a rotation matrix is its *transpose*
  - Because: the opposite of rotating by θ is rotating by $-\theta$, e.g.

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_x{}^{-1}(\theta) = R_x(-\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\theta) & -\sin(-\theta) & 0 \\ 0 & \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & sin\theta & 0 \\ 0 & -sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= R_x{}^T(\theta)$$

Inverse of a transformation matrix

- Rule 2: the inverse of a translation matrix is the same matrix with the signs on the translation components reversed
  - Because: the opposite of travelling t units in one direction is travelling t units in the opposite direction

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse of a transformation matrix

- Rule 3: the inverse of a scale matrix is a scale matrix with the reciprocal scale factors
  - Because: the opposite of making something s times bigger is making is s times smaller

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse of a transformation matrix

- Rule 4: the inverse of a matrix product is the product of the inverse matrices, ordered in reverse

$$(\boldsymbol{AB})^{-1}\boldsymbol{AB} = \boldsymbol{I}$$

$$(\boldsymbol{AB})^{-1}\boldsymbol{A}\cancel{\boldsymbol{B}\boldsymbol{B}^{-1}} = \boldsymbol{I}\boldsymbol{B}^{-1}$$

$$(\boldsymbol{AB})^{-1}\boldsymbol{A} = \boldsymbol{B}^{-1}$$

$$(\boldsymbol{AB})^{-1}\cancel{\boldsymbol{A}\boldsymbol{A}^{-1}} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$$

$$(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$$

Inverse of a transformation matrix

Example: inverting a combined rotation and translation

$$\left(\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} s_{11} & s_{12} & s_{13} & 0 \\ s_{21} & s_{22} & s_{23} & 0 \\ s_{31} & s_{32} & s_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\right)^{-1} = \begin{pmatrix} s_{11} & s_{12} & s_{13} & 0 \\ s_{21} & s_{22} & s_{23} & 0 \\ s_{31} & s_{32} & s_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}$$

$$= \begin{pmatrix} s_{11} & s_{21} & s_{31} & 0 \\ s_{12} & s_{22} & s_{32} & 0 \\ s_{13} & s_{23} & s_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_{11} & s_{21} & s_{31} & s_{11}(-t)_x+s_{21}(-t)_y+s_{31}(-t)_z \\ s_{12} & s_{22} & s_{32} & s_{12}(-t)_x+s_{22}(-t)_y+s_{32}(-t)_z \\ s_{13} & s_{23} & s_{33} & s_{13}(-t)_x+s_{23}(-t)_y+s_{33}(-t)_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Coordinate spaces

What is a coordinate space?

Definition: a space with a coordinate system defined by an *origin* and a number of *axes* equal to the dimension of the space, allowing any point in the space to be uniquely identified as a *linear combination* of distances along the axes.

In 3D coordinate space, define the unit vectors along the x, y and z axes to be $\boldsymbol{i}, \boldsymbol{j}$ and $\boldsymbol{k}$ respectively, i.e.

$$\boldsymbol{i} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \boldsymbol{j} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \boldsymbol{k} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Then any point in the space can be written as $a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k}$

Basis vectors

Definition: a set of *linearly independent* vectors $\boldsymbol{v}_1 \ldots \boldsymbol{v}_n$ in n-dimensional space form a basis for that space if any vector in that space can be expressed uniquely as a *linear combination* of the vectors $\boldsymbol{v}_i$:

$$\boldsymbol{x} = a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \ldots + a_n\boldsymbol{v}_n$$

where the coefficients a_i are the *coordinates* of $\boldsymbol{x}$

Linear independence means that $a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \ldots + a_n\boldsymbol{v}_n = \boldsymbol{0}$ if and only if $a_1 = a_2 = \ldots = a_n = 0$.

In fact: any set of n linearly independent vectors form a basis for the space.

- Therefore, the vectors $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ on the previous slide form a basis for 3D space.
- So do the vectors $\begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$, $\begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix}$ and $\begin{pmatrix} -2 \\ 1 \\ 4 \end{pmatrix}$ (proof here).

Orthonormal basis vectors

Definition: an *orthonormal basis* is a set of basis vectors that are *orthogonal* and *unit length.*

This means that:

- The coordinates are *uncoupled*, so that any given coordinate of a vector $\boldsymbol{x}$ can be determined solely from the coefficient and the corresponding basis vector.
  - Displacement along one basis vector does not cause any displacement along any of the others.
- Each coordinate of $\boldsymbol{x}$ is the signed displacement in the direction of the corresponding basis vector, which can be computed using the dot product.

Properties of a coordinate space

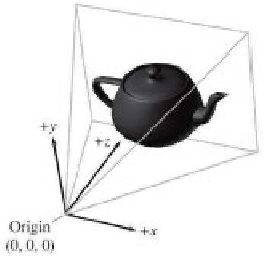
A coordinate system has:

- A set of axes (orthonormal basis) -> directions
- An origin -> position

… relative to what?!

Some common coordinate spaces

- **World space:** establishes a *global* reference frame for all other coordinate reference frames.
  - Covers the whole area/volume in which the action is currently taking place
  - Directions are fixed for all objects: e.g. north, south, east, west
- **Object space:** the *local* coordinate space associated with a particular object.
  - Origin is the object's centre of mass, root joint etc.
    - May have several nested/hierarchical spaces for different components of the model
  - Useful for applying rotation/scale
  - Directions are relative to each object: e.g. left, right, up, down



- **Camera space:** the object space associated with the viewpoint used for rendering.
  - Convention: left-handed with viewing direction along the positive z axis from the origin (camera position).
- **Screen space:** the 2D space onto which the camera space view is *projected*.

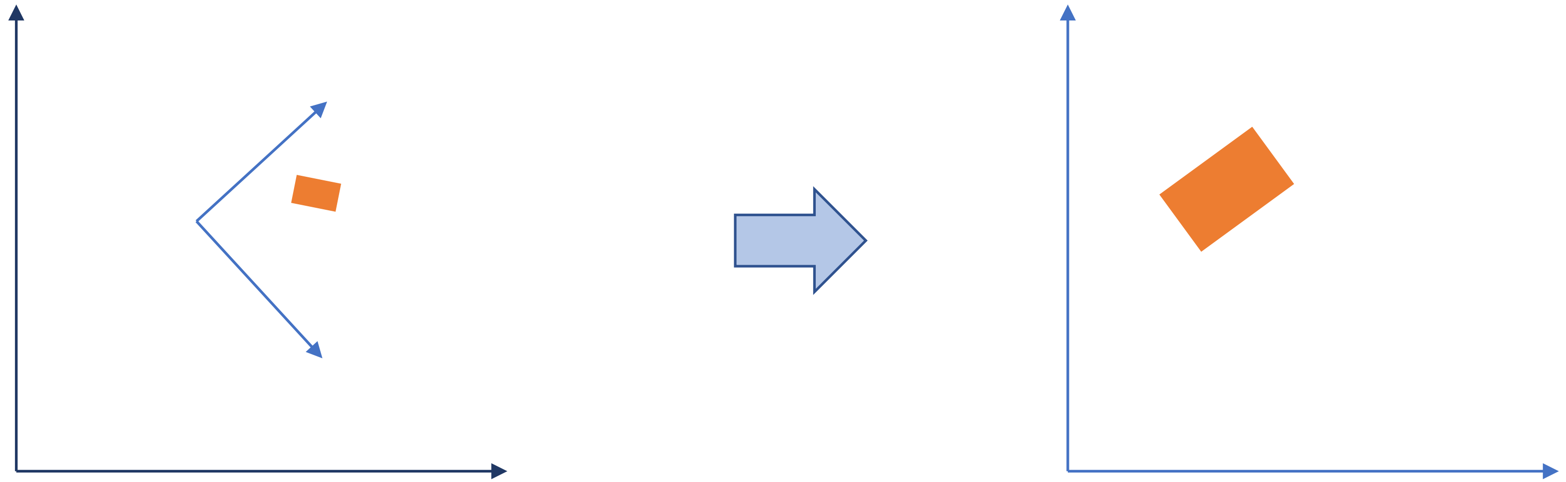
Transforming between coordinate spaces

AKA expressing points known in one coordinate space in a frame of reference relative to another, e.g.

- Individual vertices of an object are probably stored in object space, with the object's overall transform specified in world space
- To find collisions between two objects, we need both sets of vertices in the same space
  - Either transform both to world space, or one object to the other object's space (via the world)
- To render the objects, we need to know their vertex positions in camera space (via world space).

Transforming between coordinate spaces

Duality between describing a point in a different coordinate space, and applying a transformation to the point:

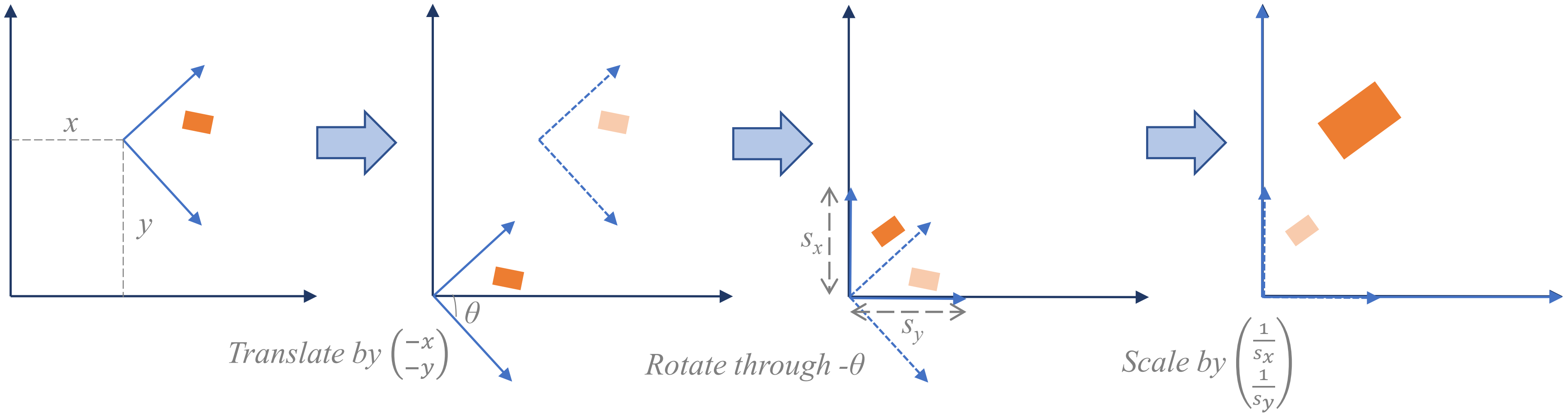


Transforming a point to a new coordinate space = transforming the new space to the old

i.e. applying the *inverse* of the new space's transform in the old space

Transforming between coordinate spaces

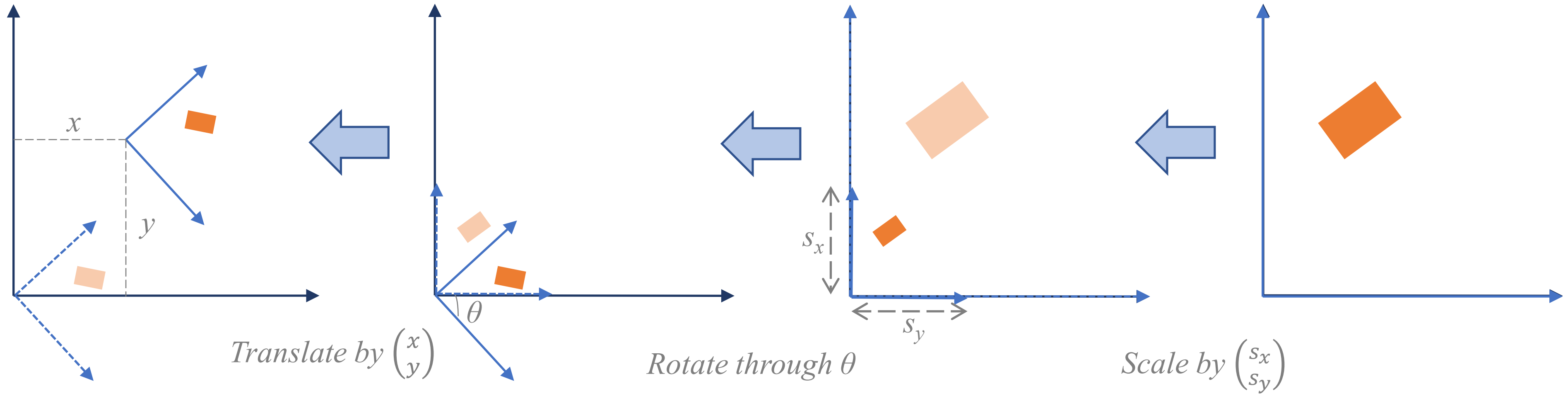
World space to local space: translate (to the world space origin), rotate, scale



i.e. we're performing the *opposite* transformation to the one that describes the local space in world coordinates

Transforming between coordinate spaces

Local space to world space: scale, rotate, translate



i.e. we're performing the *same* transformation as the one that describes the local space in world coordinates… this is just how we move objects (models) around the world!

Matrices and coordinate space transforms

Remember that a matrix describes a linear mapping:

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} m_{11}x + m_{12}y + m_{13}z \\ m_{21}x + m_{22}y + m_{23}z \\ m_{31}x + m_{32}y + m_{33}z \end{pmatrix}$$

Applied to the standard basis vectors:

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} m_{11} \\ m_{21} \\ m_{31} \end{pmatrix}$$

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} m_{12} \\ m_{22} \\ m_{32} \end{pmatrix}$$

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} m_{13} \\ m_{23} \\ m_{33} \end{pmatrix}$$

Matrices and coordinate space transforms

Theorem: the columns of a transformation matrix $\boldsymbol{M}$ can be interpreted as basis vectors of the space that $\boldsymbol{M}$ transforms to.

Since any vector $\boldsymbol{x}$ can be written as a linear combination of $\boldsymbol{i}, \boldsymbol{j}$ and $\boldsymbol{k}$:

$$\boldsymbol{x} = a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{j}$$

$$\boldsymbol{Mx} = \boldsymbol{M}(a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k})$$

$$= \boldsymbol{M}(a\boldsymbol{i}) + \boldsymbol{M}(b\boldsymbol{j}) + \boldsymbol{M}(c\boldsymbol{k})$$

$$= a(\boldsymbol{Mi}) + b(\boldsymbol{Mj}) + c(\boldsymbol{Mk})$$

$$= a\begin{pmatrix} m_{11} \\ m_{21} \\ m_{31} \end{pmatrix} + b\begin{pmatrix} m_{12} \\ m_{22} \\ m_{32} \end{pmatrix} + c\begin{pmatrix} m_{13} \\ m_{23} \\ m_{33} \end{pmatrix}$$

Note: we've already been using this fact to construct the transformation matrices (e.g. rotation)!

Tip: visualise the kind of transformation by extracting the basis vectors and comparing them to the original axes.

Example: generalised camera coordinates

Simple camera – recap:

- Camera (COP) oriented along the (positive) world-space z-axis
- Direction of view along the negative z-axis
- Vertically aligned with the positive y-axis

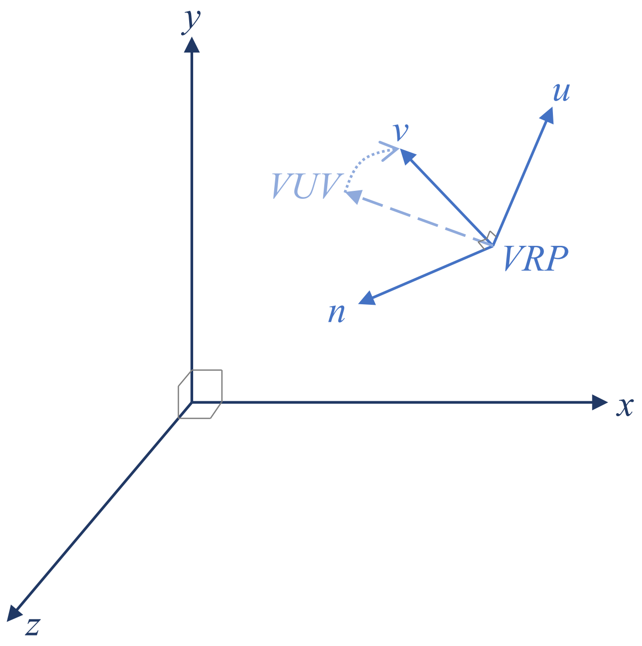
Example: generalised camera coordinates

Generalised camera – *viewing coordinate (VC) system* parameters:

- **View reference point (VRP):** the origin (point) of the VC system in world space
  - The point with respect to which the COP and view plane are defined
- **View plane normal (VPN):** direction vector specifying the positive z-axis of the VC system in world space
  - Direction the camera is pointing
- **View up vector (VUV):** direction vector used to define the positive y-axis of the VC system in world space
  - The VC y-axis is formed by projecting the VUV onto a plane perpendicular to the VPN, passing through the VRP

Example: generalised camera coordinates

Let the x-, y- and z-axes of the viewing coordinates be referred to as u, v and n respectively:



Let $\boldsymbol{n}$ be a unit vector in the direction of the VPN:

$$\boldsymbol{n} = \frac{\boldsymbol{VPN}}{\|\boldsymbol{VPN}\|}$$

Let $\boldsymbol{u}$ be a unit vector in the direction of the u-axis of the viewing coordinates. To form a left-handed system,

$$\boldsymbol{u} = \frac{\boldsymbol{n} \times \boldsymbol{VUV}}{\|\boldsymbol{n} \times \boldsymbol{VUV}\|}$$

Finally, to obtain the unit vector $\boldsymbol{v}$ in the direction of the v-axis,

$$\boldsymbol{v} = \boldsymbol{u} \times \boldsymbol{n}$$

Example: generalised camera coordinates

Now let $\boldsymbol{M}$ be the 4x4 matrix that maps world coordinate space into viewing coordinate space, partitioned into a rotational part, $\boldsymbol{R}$, and translation vector $\boldsymbol{t}$:

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{pmatrix}$$

The vectors $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{n}$ must be rotated by $\boldsymbol{R}$ into the unit basis vectors of VC space:

$$\boldsymbol{Ru} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \boldsymbol{Rv} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \boldsymbol{Rn} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

That is,

$$\boldsymbol{R}(\boldsymbol{u} \quad \boldsymbol{v} \quad \boldsymbol{n}) = \boldsymbol{I}$$

Since $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{n}$ are orthonormal, $\boldsymbol{R} = (\boldsymbol{u} \quad \boldsymbol{v} \quad \boldsymbol{n})^T = \begin{pmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ n_1 & n_2 & n_3 \end{pmatrix}$ *(explanation here).*

Example: generalised camera coordinates

Similarly, the VRP must be transformed into the origin of the VC space. If the position of the VRP in world space is given by $\boldsymbol{q}$, then

$$\boldsymbol{M}\begin{pmatrix}\boldsymbol{q}\\1\end{pmatrix}=\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

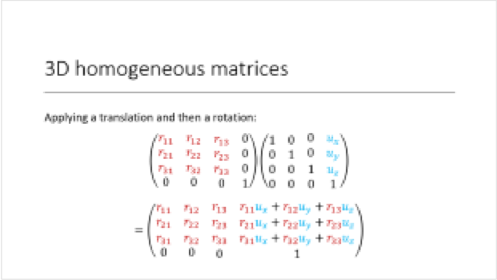
Substituting M for its partitioned form,

$$\begin{pmatrix}\boldsymbol{R} & \boldsymbol{t}\\0 & 1\end{pmatrix}\begin{pmatrix}\boldsymbol{q}\\1\end{pmatrix}=\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\Rightarrow \boldsymbol{Rq}+\boldsymbol{t}=\boldsymbol{0}\Rightarrow \boldsymbol{t}=-\boldsymbol{Rq}$$

Example: generalised camera coordinates

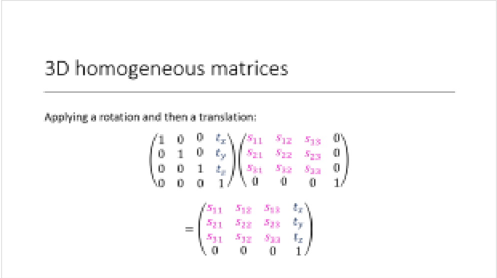
Putting everything together, we get

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{u} & \\ \boldsymbol{v} & -\boldsymbol{R}\boldsymbol{q} \\ \boldsymbol{n} & \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} u_1 & u_2 & u_3 & -\boldsymbol{u}\cdot\boldsymbol{q} \\ v_1 & v_2 & v_3 & -\boldsymbol{v}\cdot\boldsymbol{q} \\ n_1 & n_2 & n_3 & -\boldsymbol{n}\cdot\boldsymbol{q} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



In addition, the inverse (which transforms from viewing coordinates back to world coordinates) can be written as:

$$\boldsymbol{M}^{-1} = \begin{pmatrix} \boldsymbol{R}^T & \boldsymbol{q} \\ 0 & 1 \end{pmatrix}$$



More on rotations

Pros and cons of matrices

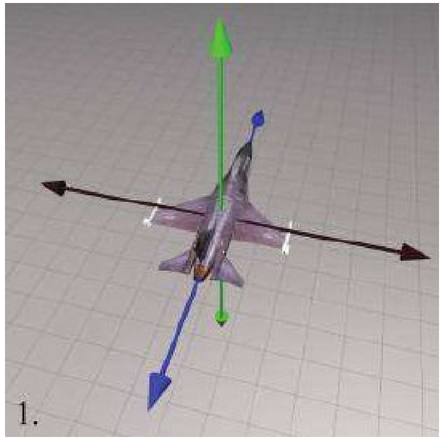
Pros	Cons
Explicit/"brute force" representation: can be applied directly to vectors	Take up more memory than is really needed for the information stored
Commonly used by graphics APIs	Not intuitive for humans to use
Concatenation of multiple transforms in a single matrix	Can easily be ill-formed: • Scale, skew, reflection or projection matrices aren't orthogonal • Bad input data, e.g. from mocap • Floating point errors (from successive incremental changes): *matrix creep* requires re-orthogonalisation
The opposite transform is given by the inverse, which is relatively straightforward to compute	

Other ways to represent rotation: Euler angles

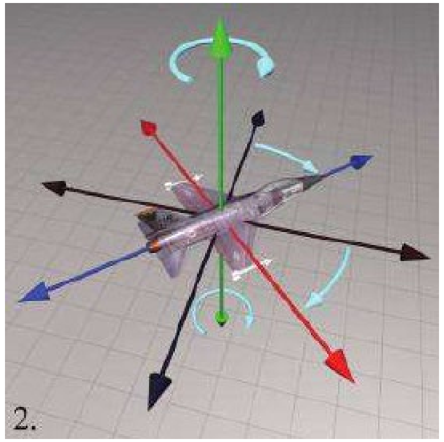
Define an angular displacement as a *sequence of three rotations* about three mutually perpendicular axes (usually x, y, z).

- Can be applied in any order – must be specified.
- Many variations on conventions/nomenclature, e.g. *yaw-pitch-roll*, or *heading-pitch-bank*
- Rotations occur about the body (local space) axes, which change after each rotation…
- Equivalent to a fixed-axes system provided that the rotations are performed in the opposite order.
- Original (symmetric) system: first and last rotations are about the same axis
- Sensible order:
  - First about the vertical axis (y)
  - Second about the body lateral axis (x)
  - Third about the body longitudinal axis (z)

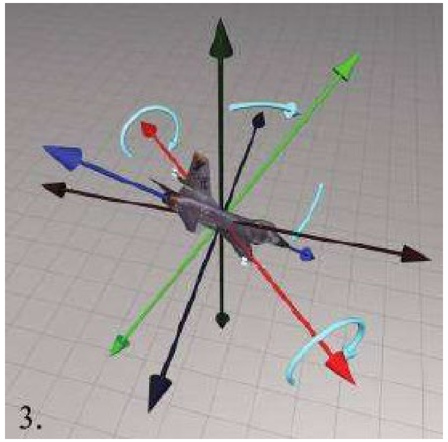
Euler angles example



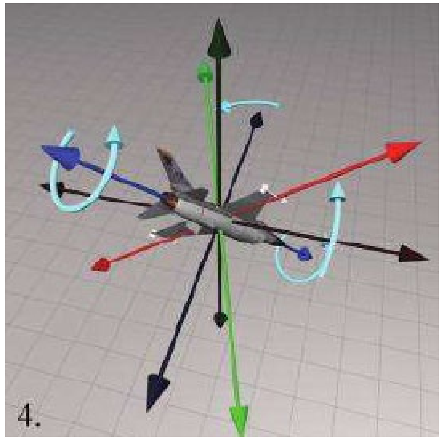
Initial orientation



Heading rotation
(vertical / y-axis)



Pitch rotation
(lateral / x-axis)



Bank rotation
(longitudinal / z-axis)

"3D Math Primer for Graphics and Game Development" (2nd Ed), figures 8.4-8.7

Euler angles and aliasing

Problem: different angles can give the same result

- Adding a multiple of 360° changes the numbers but not the rotation
- The angles are not completely independent of each other
    - e.g. pitching down 135° = heading 180°, pitching down 45°, banking 180°

Gimbal lock

(Partial) **solution:** restrict range of angles to a *canonical set*, e.g. (-180°, +180°] for heading/bank and (-90°, +90°] for pitch.

- But still: 45° right then 90° down = down 90° then bank/twist 45°
- Generally: an angle of ±90° for the second rotation causes the first and third rotations to be about the same axis

Additional restriction: assign all rotation about the vertical axis to the first (heading) rotation, leaving the last (bank) at zero.

Interpolating Euler angles

Standard linear interpolation (LERP):

$$\Delta\theta = \theta_1 - \theta_0$$
$$\theta_t = \theta_0 + t\Delta\theta$$

- Tends to choose the "long way round", even within canonical ranges, e.g. between -170° and +170°
  - Solution: wrap to find the shortest arc by adding/subtracting the appropriate multiple of 360°
- Gimbal lock causes sudden changes of orientation (angular velocity is not constant)
  - Cannot be eliminated, but can work around by choosing appropriate rotation orders for each scenario
  - https://www.youtube.com/watch?v=zc8b2Jo7mno

Pros and cons of Euler angles

Pros	Cons
More intuitive to visualise (?)	The representation for a given orientation is not unique • Angles are cyclical and not mutually independent
Smallest possible representation – no wasted space	Interpolation is problematic • Gimbal lock
Can be compressed if necessary: angle values are larger than the sine/cosine values stored in matrices, so require less precision	Need to be converted to matrices to use
Any set of three numbers is valid (will produce a valid rotation)	

Other ways to represent rotation: Quaternions

- Avoids problems with discontinuities inherent in using only 3 values to represent 3D rotations by using 4 values:
  - A scalar component, w, as well as a 3D vector component, $\boldsymbol{v} = [x \quad y \quad z]$
  - Not to be confused with homogeneous coordinates!
  - Row and column formats are interchangeable – they don't interact with matrices
- Based on *Euler's rotation theorem:* any 3D angular displacement can be accomplished by a <u>single</u> rotation through an angle θ about a carefully chosen axis $\hat{\boldsymbol{n}}$.
  - NB other methods – *axis-angle form* and *exponential map* – use these values directly.

Quaternion properties and operations

Encode the axis and angle of rotation as:

$$[w \quad \boldsymbol{v}] = \left[cos\left(\frac{\theta}{2}\right) \quad sin\left(\frac{\theta}{2}\right)\hat{\boldsymbol{n}}\right]$$

- **Negation**: $-\boldsymbol{q} = -[w \quad (x \quad y \quad z)] = [-w \quad (-x \quad -y \quad -z)]$
  $= -[w \quad \boldsymbol{v}][-w \quad -\boldsymbol{v}]$
  - But – it doesn't really do anything! $\boldsymbol{q}$ and $-\boldsymbol{q}$ describe the same angular displacement (e.g. add 360° to θ).
- **Identity**: $[1\ \boldsymbol{0}]$ and (geometrically) $[-1\ \boldsymbol{0}]$
  - Complete rotation about any axis
- **Magnitude**: $\|\boldsymbol{q}\| = \|[w \quad (x \quad y \quad z)]\| = \sqrt{w^2 + x^2 + y^2 + z^2}$
  $= \|[w \quad \boldsymbol{v}]\| = \sqrt{w^2 + \|\boldsymbol{v}\|^2}$
  $= \sqrt{cos^2\left(\frac{\theta}{2}\right) + \left(sin\left(\frac{\theta}{2}\right)\|\hat{\boldsymbol{n}}\|\right)^2} = \sqrt{cos^2\left(\frac{\theta}{2}\right) + sin^2\left(\frac{\theta}{2}\right)} = 1$

$cos^2x + sin^2x \ \equiv 1$

Quaternion inverse and multiplication

- **Conjugate**: $\boldsymbol{q}^* = [w \quad \boldsymbol{v}]^* = [w \quad -\boldsymbol{v}]$
- **Inverse**: $\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\|\boldsymbol{q}\|} = \boldsymbol{q}^*$
  - Negating the rotation axis flips the positive rotation direction
  - Negating the angle is geometrically equivalent, but not mathematically
- **Multiplication**: $\boldsymbol{q_1 q_2} = [w_1 \quad \boldsymbol{v}_1][w_2 \quad \boldsymbol{v}_2]$
  $= [w_1 w_2 - \boldsymbol{v}_1 \cdot \boldsymbol{v}_2 \quad w_1 \boldsymbol{v}_2 + w_2 \boldsymbol{v}_1 + \boldsymbol{v}_1 \times \boldsymbol{v}_2]$
  - Properties:
    - Associative but not commutative
    - $\|\boldsymbol{q}_1 \boldsymbol{q}_2\| = \|\boldsymbol{q}_1\| \|\boldsymbol{q}_2\| = 1$
    - The inverse of a quaternion product is equal to the product of the inverses in reverse order,
      $(\boldsymbol{q}_1 \boldsymbol{q}_2 \dots \boldsymbol{q}_n)^{-1} = \boldsymbol{q_n}^{-1} \boldsymbol{q_{n-1}}^{-1} \dots \boldsymbol{q}_1^{-1}$

AKA *Hamilton product*

Applying a quaternion to a point

"Extend" a point (x, y, z) into quaternion space by defining the quaternion $\boldsymbol{p} = [0 \quad (x \quad y \quad z)]$.
Note: in general, $\boldsymbol{p}$ is not a valid rotation as it can have any magnitude.

We can rotate the point $\boldsymbol{p}$ about the axis $\hat{\boldsymbol{n}}$ of a rotation quaternion $\boldsymbol{q} = \left[cos\left(\frac{\theta}{2}\right) \quad sin\left(\frac{\theta}{2}\right)\hat{\boldsymbol{n}}\right]$ by performing the multiplication

$$\boldsymbol{p}' = \boldsymbol{q}\boldsymbol{p}\boldsymbol{q}^{-1}$$

- Can be verified by conversion to a matrix for conversion about an arbitrary axis
- Uses about the same number of operations
- **Multiple rotations**: rotating $\boldsymbol{p}$ first by a quaternion $\boldsymbol{a}$ and then by another, $\boldsymbol{b}$, is equivalent to performing a single rotation by the quaternion product $\boldsymbol{ba}$:

$$\begin{aligned}\boldsymbol{p}' &= \boldsymbol{b}(\boldsymbol{a}\boldsymbol{p}\boldsymbol{a}^{-1})\boldsymbol{b}^{-1} \\ &= (\boldsymbol{ba})\boldsymbol{p}(\boldsymbol{ba})^{-1}\end{aligned}$$

Quaternion difference and exponentiation

- **Difference:** given orientations $\boldsymbol{a}$ and $\boldsymbol{b}$, the angular displacement $\boldsymbol{d}$ that rotates from $\boldsymbol{a}$ to $\boldsymbol{b}$ is given by

$$\boldsymbol{da} = \boldsymbol{b}$$
$$\boldsymbol{d} = \boldsymbol{ba}^{-1}$$

- **log:** $\log\boldsymbol{q} = \log([cos\alpha \quad \hat{\boldsymbol{n}}sin\alpha]) \equiv [0 \quad \alpha\hat{\boldsymbol{n}}]$, with $\alpha = \frac{\theta}{2}$
- **Exponentiation**: $\boldsymbol{q}^t =$ "$\boldsymbol{q}$ multiplied by itself t times" $= \exp(t\log\boldsymbol{q})$
  - As t varies from 0 to 1, $\boldsymbol{q}^t$ varies from $[1\ \boldsymbol{0}]$ to $\boldsymbol{q}$
    - Allows extraction of a "fraction" of an angular displacement
  - $\boldsymbol{q}^2$ represents twice the angular displacement of $\boldsymbol{q}$
    - Always uses the shortest arc; cannot represent multiple spins

Quaternion interpolation

Spherical linear interpolation (SLERP):

$$slerp(\boldsymbol{q}_0, \boldsymbol{q}_1, \mathrm{t}) = (\boldsymbol{q}_0 \boldsymbol{q}_1{}^{-1})^t \boldsymbol{q}_0$$

- Algebraic/theoretical form of computation
  - Actually use a mathematically equivalent, but computationally more efficient, form…

To be continued…

Pros and cons of quaternions

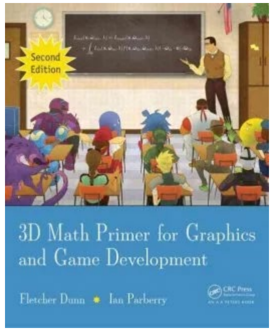
Pros	Cons
Only four values to store	One more than Euler angles • Component values do not interpolate smoothly, so harder to compress
Only representation that provides smooth interpolation	Can become invalid (from bad input or rounding errors)
Fast concatenation and inversion	Least intuitive representation
Fast conversion to and from matrix form	

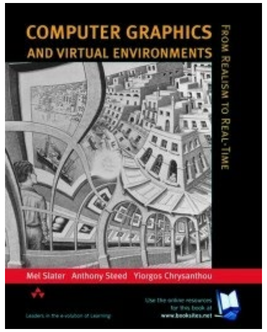
References

Some useful reading material

Books

- “3D Math Primer for Graphics and Game Development” (2^{nd} Ed)
  Fletcher Dunn and Ian Parberry, CRC Press
    - Chapters 3-6 and 8
    - Some images taken from here!
- “Computer Graphics and Virtual Environments”
  Mel Slater, Anthony Steed and Yiorgos Chrysanthou, Addison Wesley
    - Chapters 2, 5 and 7





Websites

- [ScratchAPixel]
- [Wolfram MathWorld]