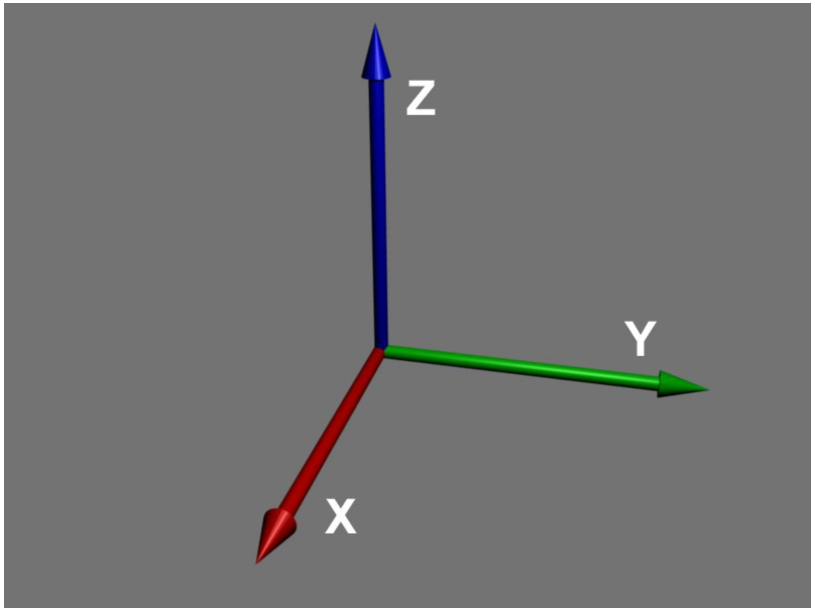
Animation Systems:
2. Basics

Charles A. Wüthrich
CogVis/MMC, Faculty of Media
Bauhaus-University Weimar

The 3D space

- Remember from CG?
  - We had a 3D space, and
  - Right handed axes, with their units
- Of course one could choose also a left-handed coordinate system
- Further on, remember that one could make coincide the x axis with the x axis of the screen, and the y axis with the UP or DOWN direction of the screen side
- Which one one uses is indifferent, as long as it is consistent throughout



Transformations

- Remember we had homogenous coordinates, with

$$[x\ y\ z] \rightarrow [x\ y\ z\ 1]$$
$$[a\ b\ c\ d] \rightarrow [a/d\ b/d\ c/d]$$

- And basic transformations:
  - Translations
$$T = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
  - Scaling
$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformations

– Rotations:

$$R_x(\vartheta)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\vartheta & -\sin\vartheta & 0\\ 0 & \sin\vartheta & \cos\vartheta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\qquad R_y(\vartheta)=\begin{bmatrix}\cos\vartheta & 0 & \sin\vartheta & 0\\ 0 & 1 & 0 & 0\\ -\sin\vartheta & 0 & \cos\vartheta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

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

– In general, we would have:

$$T=\begin{bmatrix}a_{11} & a_{12} & a_{12} & a_{14}\\ a_{21} & a_{22} & a_{23} & a_{24}\\ a_{31} & a_{32} & a_{33} & a_{34}\\ 0 & 0 & 0 & 1\end{bmatrix}$$

, where

$$\begin{array}{c}\text{rotation} \qquad\qquad\quad \text{translation}\\ \begin{bmatrix}a_{11} & a_{12} & a_{12} & a_{14}\\ a_{21} & a_{22} & a_{23} & a_{24}\\ a_{31} & a_{32} & a_{33} & a_{34}\\ 0 & 0 & 0 & 1\end{bmatrix}\end{array}$$

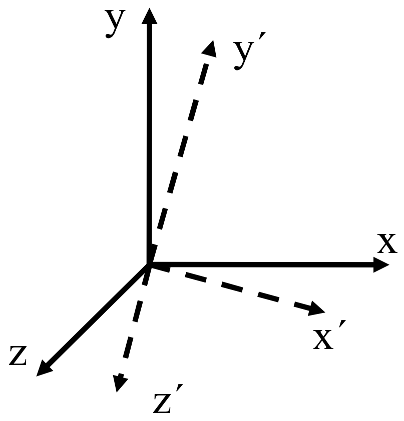
Multiple transformations

- Advantage of transformation matrices: one can combine them by simply multiplying the corresponding matrices

$$P' = M_1 P,\ P'' = M_2 P' \quad \Rightarrow P'' = M_1 M_2 P$$

- This way one can precompute once and for all the transformation matrix and apply it to all points to be transformed
- Note: matrix multiplication is non commutative

Rotating axes to a desired orientation

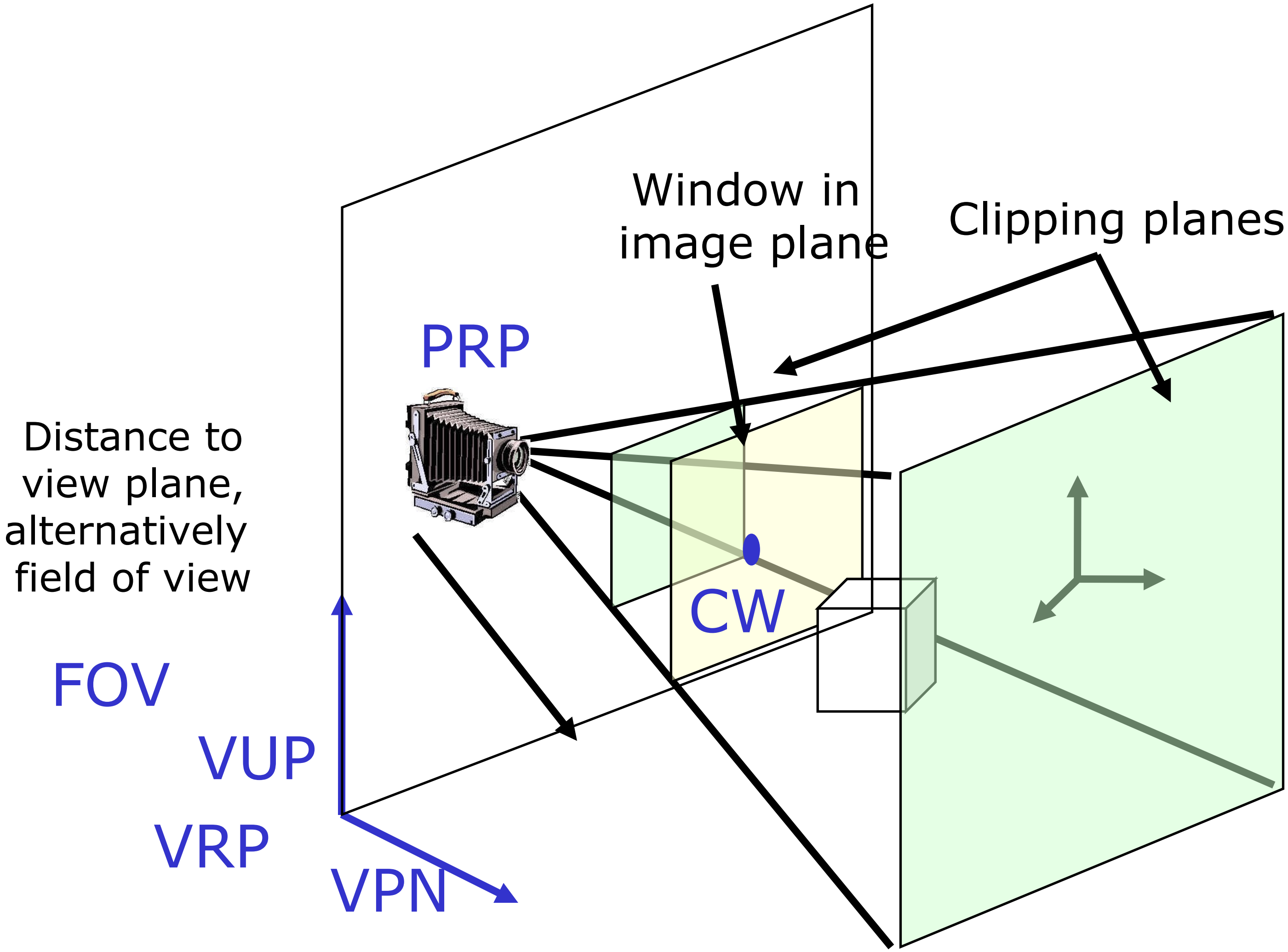


- Problem: Given a coordinate system $\underline{x}\ \underline{y}\ \underline{z}$ rotate it to a desired orientation so it coincides with $\underline{x}´\underline{y}´\underline{z}´$
- This is easy to solve: one has to find a 3x3 matrix M so that
  `$\underline{x}´=M\underline{x}$, $\underline{y}´=M\underline{y}$, $\underline{z}´=M\underline{z}$

  thus $M=\begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix}$, so $M=\begin{bmatrix} x'_x & y'_x & z'_x & 0 \\ x'_y & y'_y & z'_y & 0 \\ x'_z & y'_z & z'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

  transforms an object in the xyz coords into the coords x´y´z´
- Note: x'_x=length of projection of x´on x

Camera description



Perspective projection (to screen)

- The transformation
  $P(x,y,z) \rightarrow P_p(x_p,y_p,0)$ is performed by multiplying with the matrix M_{per}:

$$P_p = M_{per} P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 - z/d \end{bmatrix}$$

=> perspective transforms are 4x4 matrices too

Representing object orientation

- How do I represent best the position and orienta-tion of a object in space so as to „move" it in time?
- A transformation matrix

$$T = \begin{array}{c} \begin{array}{cccc} & \text{rotation} & & \text{translation} \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & a_{12} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

will always be the result of the successive application of a 3x3 rotation matrix and of a translation (if the body is rigid)

Representing object orientation

- Thus,

$$T=\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & b_{14} \\ 0 & 1 & 0 & b_{24} \\ 0 & 0 & 1 & b_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$b_1=\frac{a_{14}}{a_{11}a_{14}+a_{12}a_{24}+a_{13}a_{34}}$$

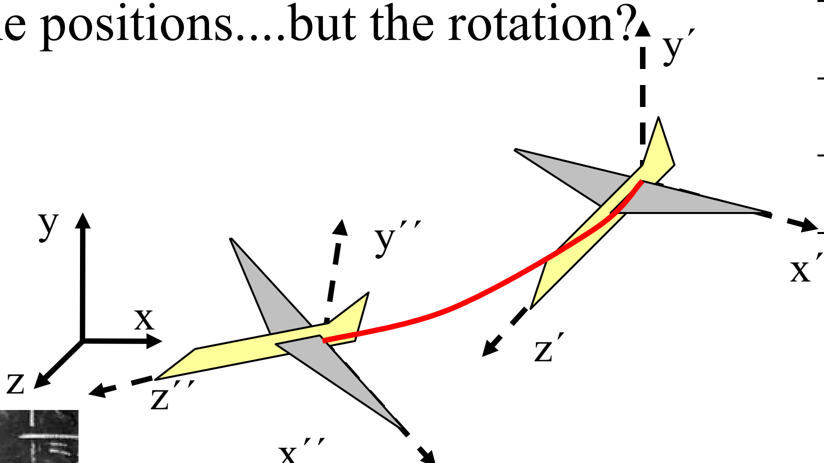
$$b_2=\frac{a_{24}}{a_{21}a_{14}+a_{22}a_{24}+a_{23}a_{34}}$$

$$b_3=\frac{a_{14}}{a_{31}a_{14}+a_{32}a_{24}+a_{33}a_{34}}$$

which means, one can consider the rotation separate from the translation to compute and anim.

Representing object orientation

- Suppose that I defined two key positions of a rigid body, and that I want to compute the equal steps between the two positions to compute the animation (each key position been defined by a Rotation-translation pair)
- For the translation part, it seems to be easy to interpolate between the positions....but the rotation?



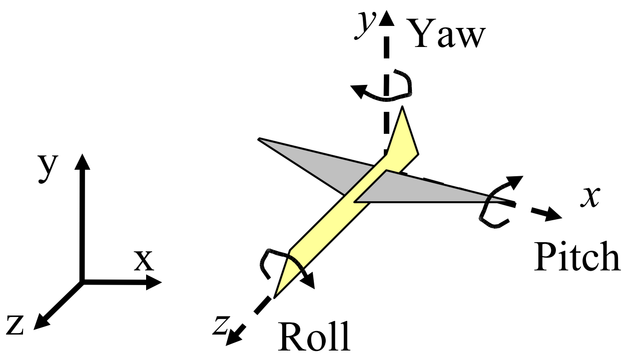
- Direct interpolation does not work, because the resulting interpolation matrices will not be normalized....
- But there ARE alternative methods to do this:
  - Fixed angle
  - Euler angle
  - Axis angle
  - Quaternions

Fixed angle representation

- Angles used to rotate around fixed axes
- One can rotate first around one main axis, then the second and then the third
- As long as one keeps always the same order, one should be fine
- But, if you apply consequently those, the second rotation will influence back the first rotation
- This effect is called *gimbal lock*
- The same problem makes interpolation between key positions a problem sometimes
- The resulting rotations will make the object swing out of the desired rotating plane

Euler angle representation

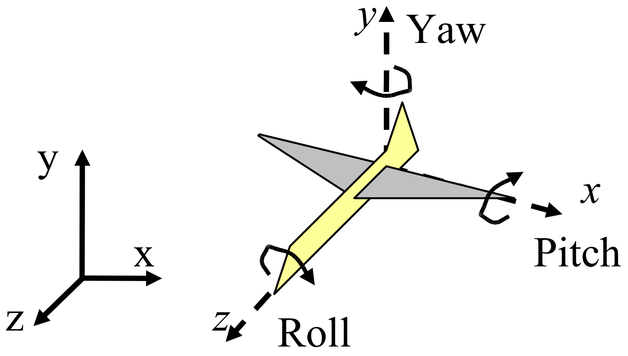
- Here the axes of rotation are on the local coordinate system of the object



- Also here, the order of the rotations is indifferent
- In fact, this method is very similar to fixed axes, and has same advantages and disadvantages
- Euler‘s rotation theorem: any orientation can be derived from another by ONE rotation around a particular axis

Euler angle representation

- Thus, given an object, any orientation can be represented with
  - An angle
  - An axis, i.e. a vector



- This can be used by using vector variations and angle intervals for computing the interpolation function
- Reasoning on vector interpolation and axis interpolation is much easier

Quaternions

- This is the better approach to do interpolation of intermediate orientations when the object has 3 DOF
- A *quaternion* is a 4-tuple of real numbers $[a,b,c,d]$.
- Equivalently, it is a pair $[s,\underline{v}]$ of a scalar s and a 3D vector $\underline{v}$.
- More, it can be defined as $w + xi + yj + zk$ (where $i^2 = j^2 = k^2 = -1$ and $ij = k = -ji$ with real w, x, y, z)

- On quaternions one defines two operations:
  - Addition: $[s_1,\underline{v}_1] + [s_2,\underline{v}_2] = [s_1+s_2,\underline{v}_1+\underline{v}_2]$
  - Multiplication: $[s_1,\underline{v}_1] \cdot [s_2,\underline{v}_2] = [s_1 \cdot s_2 - \underline{v}_1 \bullet \underline{v}_2, s_1 \cdot \underline{v}_2 + s_2 \cdot \underline{v}_1 + \underline{v}_1 \times \underline{v}_2]$
  - Note that multiplication is associative, but NOT commutative $\Rightarrow q_1 q_2 \neq q_2 q_1$

Quaternions: definitions

- Units:
  - Additive: $[0,\underline{0}]$
  - Multiplicative: $[1,\underline{0}]=[1,0,0,0]$
- Let $\underline{v}=[x,y,z]$. Inverse:
  - $q^{-1}=[s,\underline{v}]^{-1}=(1/\|q\|)^2\cdot[s,-\underline{v}]$, *where* $\|q\|=(s^2+\|\underline{v}\|)^{1/2}$
- Obviously, $qq^{-1}=[1,0,0,0]$
- A point in 3D space can be also represented as the quaternion $[0.\underline{v}]$.
  - or, alternatively, a vector from the origin
- Property: $[0,\underline{v}_1]\cdot[0,\underline{v}_2]=[0,\underline{v}_1\times\underline{v}_2]$ *iff* $\underline{v}_1\times\underline{v}_2=0$
- *Def*: Unit-length quaternion is a quaternion q such that $\|q\|=1$.
- Obviously $\forall q$, $q/\|q\|$ is a unit length quaternion

Rotating vectors through quaternions

- Consider a vector $[0,\underline{v}]$, and consider a quaternion q:
  - The rotated vector v' of v through the quaternion q is the vector
    $$v' = Rot_q(v) = q \cdot v \cdot q^{-1}$$
  - A sequence of rotations can be chained:
    $Rot_p(Rot_q(v)) = q(p \cdot v \cdot p^{-1}) \cdot q^{-1}$
    $= (q \cdot p) \cdot v \cdot (p^{-1} \cdot q)^{-1} = Rot_{pq}(v)$
  - Note that:
    $Rot^{-1}(Rot(v)) = v$

Why is it called rotation?

- The quaternion form of a rotation encodes axis-angle information.
- Let $q=[\theta,x,y,z]$ be a unit length quaternion.
- The following equation shows the unit representation of a rotation of an angle θ about the axis of rotation $\underline{v}=(x,y,z)$

$q=\mathrm{Rot}_{[\theta,(x,y,z)]}=$
$[\cos(\theta/2),\sin(\theta/2)\cdot(x,y,z)]=$
$[\cos(\theta/2),\sin(\theta/2)\cdot \underline{v}]=$

- Converting from angle and axis notation to quaternion notation involves therefore two trigonometric operations, as well as several multiplies and divisions.
- Notice that a quaternion and its negation $[-s,-\underline{v}]$ produce the same rotation (to prove it, simply write the formula here on the left for $-q$ and you will see that the negative terms will disappear)

From Euler angles to quaternions

- Converting Euler angles into quaternions is a similar process
- just have to be careful that operations are performed in correct order.
- For example, let's say that a plane in a flight simulator first performs a yaw, then a pitch, and finally a roll.
- One can represent this combined quaternion rotation as
  $q = q_{yaw}\ q_{pitch}\ q_{roll}$ where:
  $q_{roll} = [\cos (y/2), (\sin(y/2), 0, 0)]$
  $q_{pitch}=[\cos (q/2), (0, \sin(q/2), 0)]$
  $q_{yaw} = [\cos(f/2), (0, 0, \sin(f/2)]$
- The order in which the multiplications are done is important.
- Quaternion multiplication is not commutative (due to the vector cross product that's involved).
- In other words, changing the order in which you rotate an object around various axes can produce different resulting orientations, and therefore, the order is important.

From quaternions to a rotation matrix

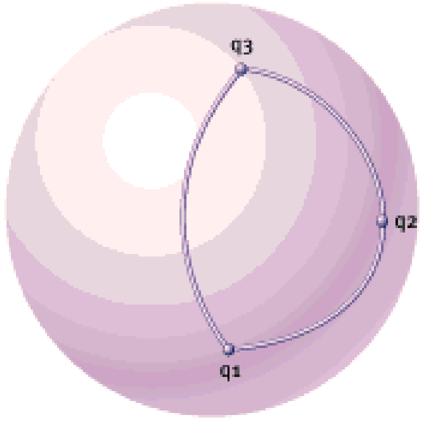
- Converting from a rotation matrix to a quaternion representation is a bit more difficult
- Conversion between a unit quaternion and a rotation matrix can be specified as

$$R_m = \begin{bmatrix} 1-2y^2-2x^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz-wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix}$$

- It's very difficult to specify a rotation directly using quaternions. It's best to store your character's or object's orientation as a Euler angle and convert it to quaternions before you start interpolating.
- It's much easier to increment rotation around an angle, after getting the user's input, using Euler angles (that is, roll = roll + 1), than to directly recalculate a quaternion.
- If the quaternions are not unit quaternions, additional multiplications and a division are required in the computation.

From quaternions to a rotation matrix

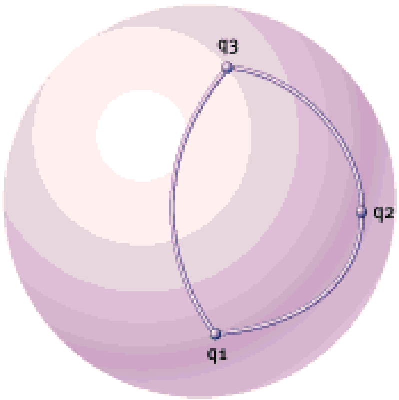
- One of the most useful aspects of quaternions is the fact that it's easy to interpolate between two quaternion orientations and achieve smooth animation.
- To demonstrate why this is so, let's look at an example using spherical rotations.
- Spherical quaternion interpolations follow the shortest path (arc) on a four-dimensional, unit quaternion sphere.
- Since 4D spheres are difficult to imagine, we'll use a 3D sphere to visualize quaternion rotations and interpolations.



Courtesy http://www.gamasutra.com

- Let's assume that the initial orientation of a vector emanating from the center of the sphere can be represented by q1 and the final orientation of the vector is q3.

From quaternions to a rotation matrix



Courtesy
http://www.gamasutra.com

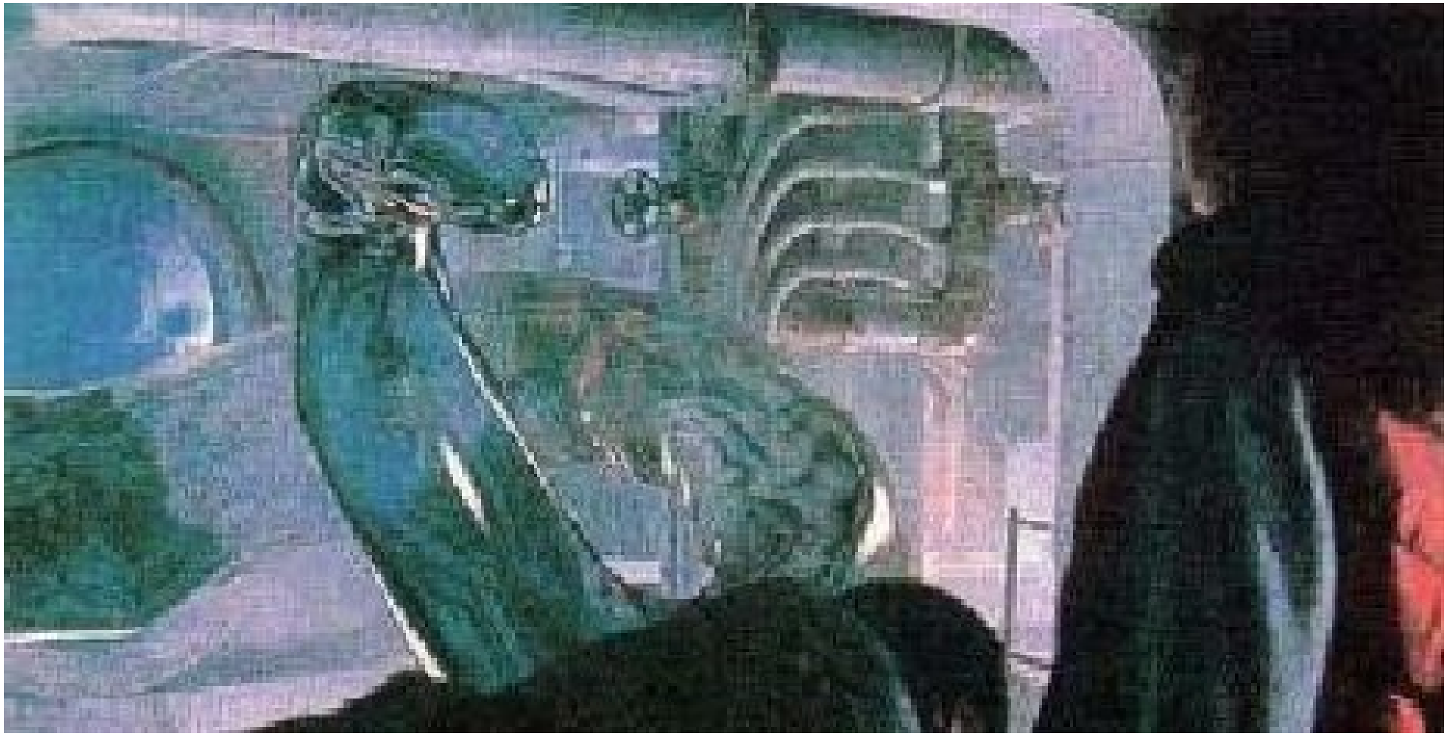
- The figure shows that if we have an intermediate position q2, the interpolation from q1→q2→q3 will not necessarily follow the same path as the q1→q3 interpolation.

- The initial and final orientations are the same, but the arcs are not.
- Quaternions simplify the calculations required when compositing rotations. For example, if you have two or more orientations represented as matrices, it is easy to combine them by multiplying two intermediate rotations.

$$R = R_2 R_1$$

- Note: R_2R_1 means rotation R_1 followed by a rotation R_2

End



+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++