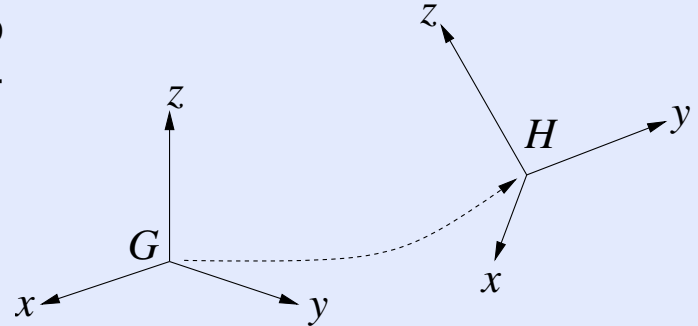


Quaternions For Pose Interpolation

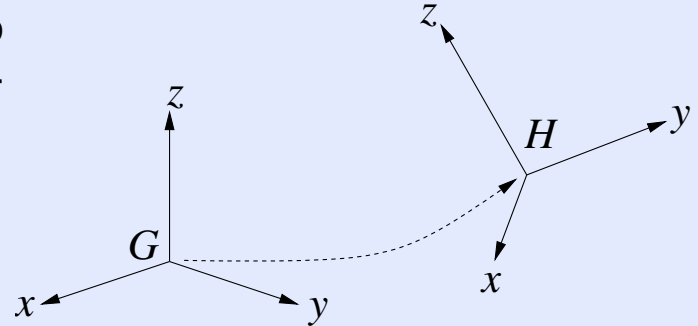
- The pose interpolation problem takes two orientations G and H as input, and produces a sequence of intermediate poses.



- Frames are given as orthonormal vectors $v_g[2]$ and and orthonormal vectors $v_h[2]$ with respect to some underlying frame F
- The goal is to produce “realistic” or believable motion

Quaternions For Pose Interpolation

- The pose interpolation problem takes two orientations G and H as input, and produces a sequence of intermediate poses.



- Frames are given as orthonormal vectors $v_g[2]$ and orthonormal vectors $v_h[2]$ with respect to some underlying frame F
- The goal is to produce “realistic” or believable motion
- Quaternions — invented by a 19th century mathematician – are a fascinating solution to this problem
- Caution: Since interpolation is not completely satisfied by two end frames, quaternions provide just “one” solution.

Ideas That Do Not Work

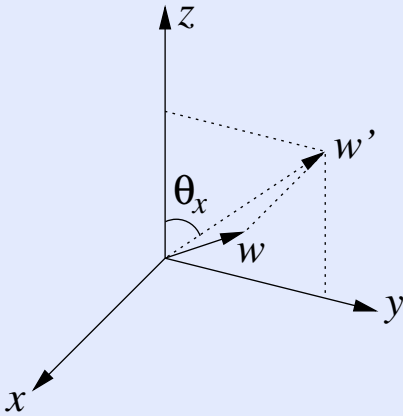
- If there were only translation (without rotation) the difficulty is reduced
- If there were rotation (and translation), we could translate the axes to frame F
- We could then consider an intermediate frame as $v_i[2]$ computed as a *convex* combination of individual vectors. For example, $v_i[1] = (1 - t)v_g[1] + tv_h[1]$ for a (time) parameter t
- Given any such vector triplet, we can align this axis with F to draw our object

Ideas That Do Not Work

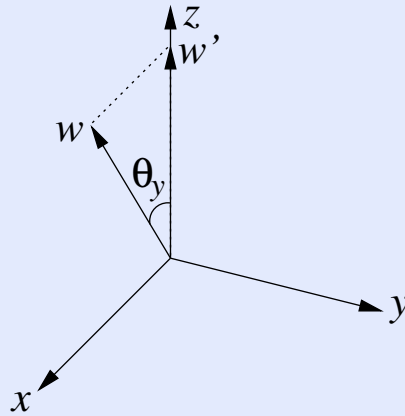
- If there were only translation (without rotation) the difficulty is reduced
- If there were rotation (and translation), we could translate the axes to frame F
- We could then consider an intermediate frame as $v_i[2]$ computed as a *convex* combination of individual vectors. For example, $v_i[1] = (1 - t)v_g[1] + tv_h[1]$ for a (time) parameter t
- Given any such vector triplet, we can align this axis with F to draw our object
- But the net resulting vector triplet is not necessarily orthogonal (e.g. $v_g[0] = [1, 1, 1]$, $v_h[0] = [8, 12, 8]$, $v_g[1] = [2, -3, 1]$, and $v_h[1] = [1, -2/3, 1]$)
- An alternative is to use Euler angles as a representation for orientation.

Using Euler Angles

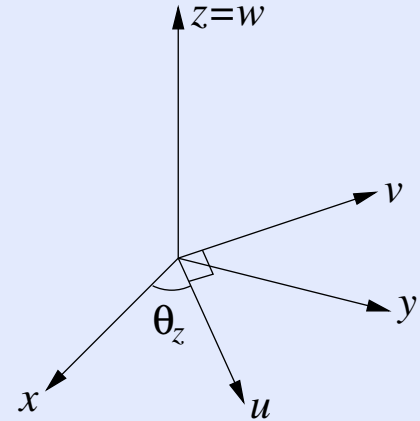
- For instance, we could re-label the x , y , and z axes ($v_g[2]$) of G as u , v and w .



(a) Rotate about x so that w lies in xz -plane.



(b) Rotate about y so that w coincides with z .



(c) Rotate about z so that u and v coincide with x and y .

- Focusing on w , we can align this axis with the z axis of the original frame F , and making u coincide with the x axis
- The net rotation matrix is $R(\Theta_g) = R(\theta_x, \theta_y, \theta_z) = R_x(\theta_x)R_y(\theta_y)R_z(\theta_z)$ and is independent of the order in which we did the rotation for aligning the two frames

Using Euler Angles

- So an orientation is represented by an axis and an angle
- We could then write $\Theta_i = (1 - t)\Theta_g + t\Theta_h$
- Incidentally given any unit vector (C_x, C_y, C_z) the matrix to align it with the z axis is

$$\begin{bmatrix} d & -\frac{C_x C_y}{d} & -\frac{C_x C_z}{d} & 0 \\ 0 & \frac{C_z}{d} & -\frac{C_y}{d} & 0 \\ C_x & C_y & C_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $(d^2 = C_z^2 + C_y^2)$

Using Euler Angles

- So an orientation is represented by an axis and an angle
- We could then write $\Theta_i = (1 - t)\Theta_g + t\Theta_h$
- Incidentally given any unit vector (C_x, C_y, C_z) the matrix to align it with the z axis is

$$\begin{bmatrix} d & -\frac{C_x C_y}{d} & -\frac{C_x C_z}{d} & 0 \\ 0 & \frac{C_z}{d} & -\frac{C_y}{d} & 0 \\ C_x & C_y & C_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $(d^2 = C_z^2 + C_y^2)$

- Unfortunately, intermediate angle Θ_i depends on frame F , and the vector relabeling algorithm

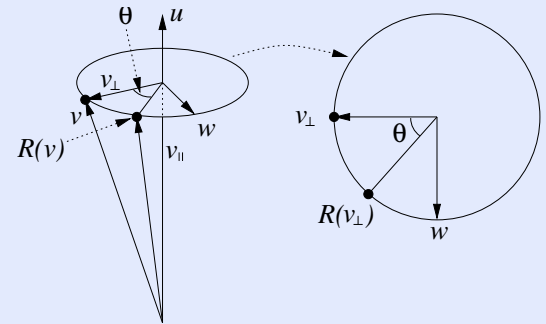
Coordinate Free Rotation

- Represent rotation by an axis \vec{u} (unit vector, still in some coordinate system) and an angle θ
- Want $R(\vec{v})$ for some point \vec{v}

Coordinate Free Rotation

- Represent rotation by an axis \vec{u} (unit vector, still in some coordinate system) and an angle θ
- Want $R(\vec{v})$ for some point \vec{v}
- Key: Write $v = v_{\parallel} + v_{\perp}$ where $v_{\parallel} = (v.u)\vec{u}$ and $v_{\perp} = \vec{v} - (v.u)\vec{u}$. Rotation is now dependent only on v_{\perp}
- Set $\vec{w} = \vec{u} \times \vec{v}_{\perp} = \vec{u} \times \vec{v}$. Note that length of \vec{w} equals length of \vec{v}_{\perp}

$$\begin{aligned}
 R(\vec{v}) &= R(\vec{v}_{\parallel} + \vec{v}_{\perp}) \\
 &= \vec{v}_{\parallel} + \cos \theta \vec{v}_{\perp} + \sin \theta (\vec{u} \times \vec{v}) \\
 &= (u.v)\vec{v} + \cos \theta (\vec{v} - (v.u)\vec{u}) + \sin \theta (\vec{u} \times \vec{v}) \\
 &= \cos \theta (\vec{v}) + (1 - \cos \theta)(v.u)\vec{u} + \sin \theta (\vec{u} \times \vec{v})
 \end{aligned}$$



- Surprising resemblance to . . . quaternions

Quaternion Basics

- Quaternions are a generalization of the 'ordinary' complex number system: Just like $\vec{v} = a + ib$, we write $\mathbf{q} = w + xi + yj + zk$ with the rules that $i^2 = j^2 = k^2 = -1$ and $ij = k$, $jk = i$, and $ki = j$.
- Complex numbers are a field over $+$ and $*$: Both operations are commutative and associative, $+$ distribute over $*$. Identity and inverse for $+$ and $*$

Quaternion Basics

- Quaternions are a generalization of the ‘ordinary’ complex number system: Just like $\vec{v} = a + ib$, we write $\mathbf{q} = w + xi + yj + zk$ with the rules that $i^2 = j^2 = k^2 = -1$ and $ij = k$, $jk = i$, and $ki = j$.
- Complex numbers are a field over $+$ and $*$: Both operations are commutative and associative, $+$ distribute over $*$. Identity and inverse for $+$ and $*$
- Quaternions are a *division ring*: multiplication is not commutative. To define $*$ write $\mathbf{q} = (w, \vec{r})$. Then $\mathbf{q}_1\mathbf{q}_2 = (w_1w_2 + \vec{r}_1 \cdot \vec{r}_2, w_1\vec{r}_2 + w_2\vec{r}_1 + \vec{r}_1 \times \vec{r}_2)$
- The *conjugate* quaternion is $\bar{\mathbf{q}} = (w, -\vec{r})$. $\mathbf{q}\bar{\mathbf{q}}$ is a scalar. The *magnitude* of \mathbf{q} is the square root of this scalar. A *unit* quaternion has unit magnitude.
- The inverse quaternion for nonzero magnitude \mathbf{q} is $\mathbf{q}^{-1} = \frac{1}{\mathbf{q}\bar{\mathbf{q}}}\mathbf{q}$
- A *pure* quaternion has zero scalar component: $\mathbf{p} = (0, \vec{r})$

Unit Quaternions And Rotation

- Given a unit quaternion $\mathbf{q} = (w, r)$ and a pure quaternion $\mathbf{p} = (0, v)$ we define the rotation operator as $R_q(p) = qpq^{-1}$
- Chugging through the algebra we find that the result is also a pure quaternion

$$(0, (w^2 - r.r)\vec{v} + 2(r.v)\vec{r} + 2w(\vec{r} \times v)) \quad (1)$$

Unit Quaternions And Rotation

- Given a unit quaternion $\mathbf{q} = (w, r)$ and a pure quaternion $\mathbf{p} = (0, v)$ we define the rotation operator as $R_q(p) = qpq^{-1}$
- Chugging through the algebra we find that the result is also a pure quaternion

$$(0, (w^2 - r.r)\vec{v} + 2(r.v)\vec{r} + 2w(\vec{r} \times v)) \quad (1)$$

- To associate this with the input for rotation θ and u , we make the association $\mathbf{q} = (\cos \theta, \sin \theta u)$ and plug it into Equation 1.

Unit Quaternions And Rotation

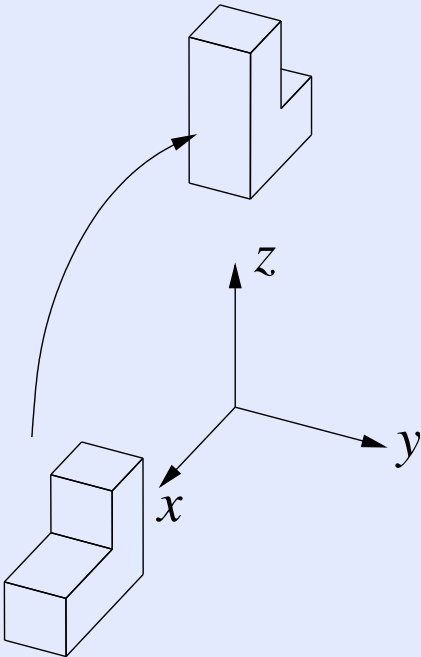
- Given a unit quaternion $\mathbf{q} = (w, r)$ and a pure quaternion $\mathbf{p} = (0, v)$ we define the rotation operator as $R_q(p) = qpq^{-1}$
- Chugging through the algebra we find that the result is also a pure quaternion

$$(0, (w^2 - r.r)\vec{v} + 2(r.v)\vec{r} + 2w(\vec{r} \times v)) \quad (1)$$

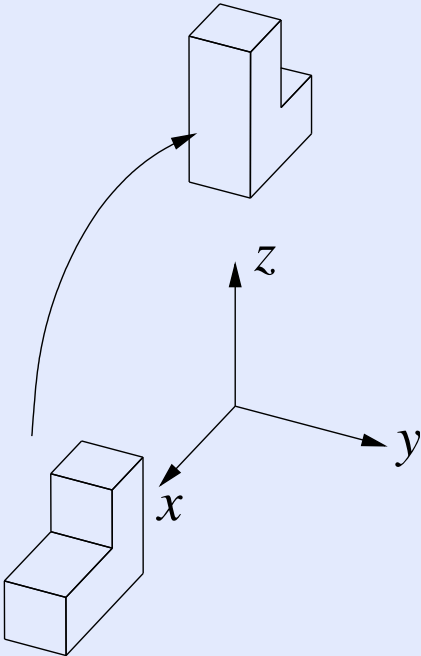
- To associate this with the input for rotation θ and u , we make the association $\mathbf{q} = (\cos \theta, \sin \theta u)$ and plug it into Equation 1.
- The result bears an uncanny similarity to coordinate-free rotation except that we are working with twice the angle

Example: Unit Quaternions And Rotation

- Consider the rotation shown in the figure

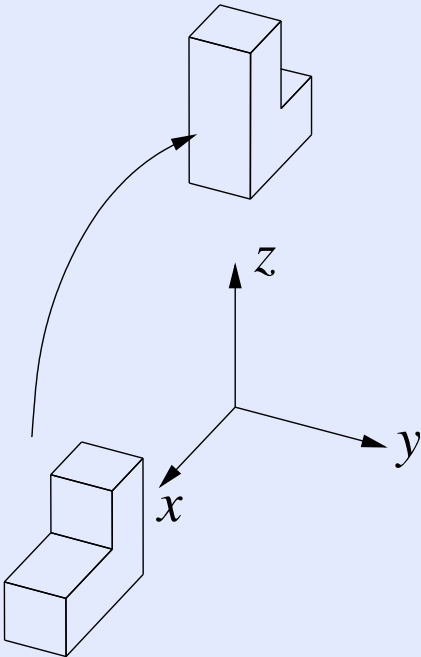


Example: Unit Quaternions And Rotation



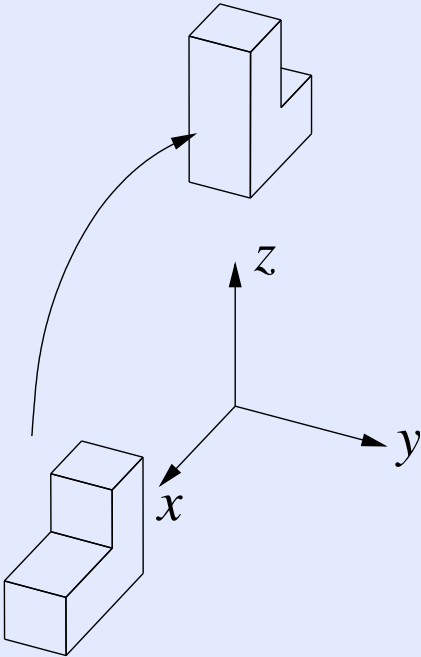
- Consider the rotation shown in the figure
- Here $\theta = -90$ and $u = (0, 1, 0)$.

Example: Unit Quaternions And Rotation



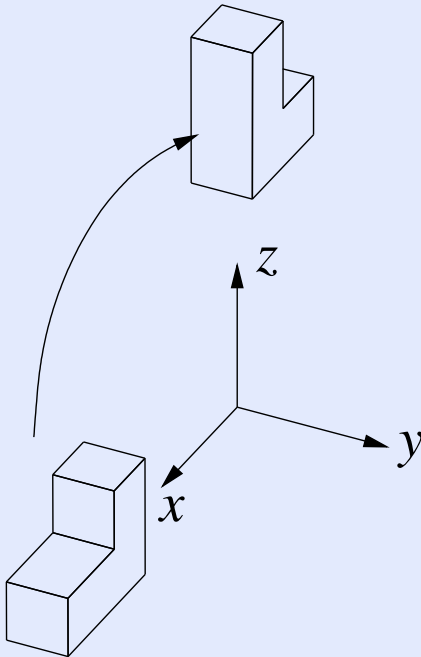
- Consider the rotation shown in the figure
- Here $\theta = -90$ and $u = (0, 1, 0)$.
- Therefore $\mathbf{q} \approx (0.707, 0, -0.707, 0)$ and

Example: Unit Quaternions And Rotation



- Consider the rotation shown in the figure
- Here $\theta = -90$ and $u = (0, 1, 0)$.
- Therefore $\mathbf{q} \approx (0.707, 0, -0.707, 0)$ and $R_{\mathbf{q}}(0, (1, 0, 0) = 0, (0, 0, 1))$ as expected

Example: Unit Quaternions And Rotation



- Consider the rotation shown in the figure
- Here $\theta = -90$ and $u = (0, 1, 0)$.
- Therefore $\mathbf{q} \approx (0.707, 0, -0.707, 0)$ and $R_{\mathbf{q}}(0, (1, 0, 0) = 0, (0, 0, 1))$ as expected

- Not only this, but we can compose rotations by multiplying quaternions. This “means” that unit quaternions are a good representation for coordinate free orientation with multiplication as the key operator

Linear Interpolation Using Quaternions

1. Given G as an orientation, construct unit quaternion q_g
2. Given H as an orientation, construct unit quaternion q_h
3. Create the convex combination q_i for a parameter t
4. Write the resulting unit quaternion as a rotation matrix, and use OpenGL to render

Quaternion Multiplication Using Matrices

- Using coordinate representations for $\mathbf{q} = w + xi + yj + zk$ we have

$$\begin{aligned}\mathbf{q}_1\mathbf{q}_2 &= (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2) \\ &+ (w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2)i \\ &+ (w_1y_2 - x_1z_2 + y_1w_2 + z_1x_2)j \\ &+ (w_1z_2 + x_1y_2 - y_1x_2 + z_1w_2)k\end{aligned}$$

- We can write $\mathbf{q}_1\mathbf{q}_2$ as the matrix product $\mathbf{q}_2L(q_1)$ or $\mathbf{q}_1R(q_2)$ if we create matrices $L(q)$ and $R(q)$ from q as below:

$$\begin{bmatrix} w_1 & z_1 & -y_1 & -x_1 \\ -z_1 & w_1 & x_1 & -y_1 \\ y_1 & -x_1 & w_1 & -z_1 \\ x_1 & y_1 & z_1 & w_1 \end{bmatrix}$$

$$\begin{bmatrix} w_2 & -z_2 & y_2 & -x_2 \\ z_2 & w_2 & -x_2 & -y_2 \\ -y_2 & -x_2 & w_2 & -z_2 \\ x_2 & y_2 & z_2 & w_2 \end{bmatrix}$$

Quaternion Multiplication Using Matrices

- We now write the rotation operator as

$$qPq^{-1} = q(P\bar{q}) = q(PR(\bar{q})) = (PR(\bar{q}))L(q) = P(R(\bar{q})L(q)) = PQ$$

where $Q = R(\bar{q})L(q)$ is

$$\begin{bmatrix} w_q & z_q & -y_q & x_q \\ -z_q & w_q & x_q & y_q \\ y_q & x_q & w_q & z_q \\ -x_q & -y_q & -z_q & w_q \end{bmatrix} \begin{bmatrix} w_q & z_q & -y_q & -x_q \\ -z_q & w_q & x_q & -y_q \\ y_q & -x_q & w_q & -z_q \\ x_q & y_q & z_q & w_q \end{bmatrix}$$

$$\begin{bmatrix} w^2 + x^2 - y^2 - z^2 & 2xy + 2wz & 2xz - 2wy & 0 \\ 2xy - 2wz & w^2 - x^2 + y^2 - z^2 & 2yz + 2wx & 0 \\ 2xz + 2wy & 2yz - 2wx & w^2 - x^2 - y^2 + z^2 & 0 \\ 0 & 0 & 0 & w^2 + x^2 + y^2 + z^2 \end{bmatrix}$$

Converting Rotation Matrices To Quaternions

- Structure of Q makes it possible to obtain the quaternion $[x, y, z, w]$ in different ways.

Converting Rotation Matrices To Quaternions

- Structure of Q makes it possible to obtain the quaternion $[x, y, z, w]$ in different ways.
- If $A, B, C,$ and D are the diagonal elements, then we have $4x^2 = A - B - C + D,$
 $4y^2 = -A + B - C + D,$ $4z^2 = -A - B + C + D$ and $4w^2 = A + B + C + D$
- Depending on which of the four quantities is the largest, we compute x or y or z or w .
 - If w is known with best precision, then we compute other quantities using $Q' = Q - Q^T$. For example, we compute x using $8wx = Q'[1][2] - Q'[2][1] \equiv Q'[9] - Q'[6]$.
 - If one of $x, y,$ or z is known with best precision, then we compute others using relevant columns in $Q'' = Q + Q^T$ (ignoring diagonal terms in Q''). Finally w is computed using Q'