Game Physics

GPR350, Fall 2019
Daniel S. Buckstein

Quaternions for Physics
Week 8

License

Quaternions

- Review of rotation matrices and their issues
- Quaternions and applications
  - Operations
  - Comparison with matrices
  - Applications

Rotation Matrices

- 3x3 matrices can be used to represent *rotations in 3D*

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

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

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

Rotation Matrices

- ***Rodrigues' rotation formula*** (axis-angle):

$$R = I + (\sin\theta)S + (1 - \cos\theta)S^2$$

where I is the 3x3 identity matrix,

θ is the angle of rotation, and

$S = \begin{bmatrix} 0 & -\hat{n}_z & \hat{n}_y \\ \hat{n}_z & 0 & -\hat{n}_x \\ -\hat{n}_y & \hat{n}_x & 0 \end{bmatrix}$, where $\hat{n}$ is the normalized axis of rotation

Rotation Matrices

- **Concatenation**:
- Also known as matrix multiplication
- ***Non-commutative***: written order matters!

$$AB \neq BA$$

- ***Associative***: chaining more than 2 matrices, does not matter which concatenation happens first

$$(AB)C = A(BC)$$

Rotation Matrices

- **Concatenation**:
- Easy way to multiply rotation matrices:
- For the matrix product $C = AB$
- For each element $C_{r,c}$ where r is the row and c is the column…
- …take the *dot product* of row r in matrix A (the left) and column c in matrix B (right)

Rotation Matrices

- **Concatenation**:
- When describing rotations, the first rotation is written on the *right*
- E.g. $R = R_0 R_1$
- In this example, the rotation R_1 will occur first
- This is not the same as $R_1 R_0$
- Easily demonstrated with real objects!

Rotation Matrices

- **Inverse**: finding the inverse of a 3x3 matrix is a time-consuming process
- PRO TIP: For *rotation matrices*, the transpose is also the inverse!!! $R^{-1} = R^T$
- How do you know if a 3x3 matrix is a rotation matrix???

Rotation Matrices

- A 3x3 matrix can be used as a rotation if its *determinant* is *equal to 1*
- **Determinant**: For a 3x3 matrix *A*

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

- The determinant is

$$\det(A) = a(ei - fh) + b(fg - di) + c(dh - eg)$$

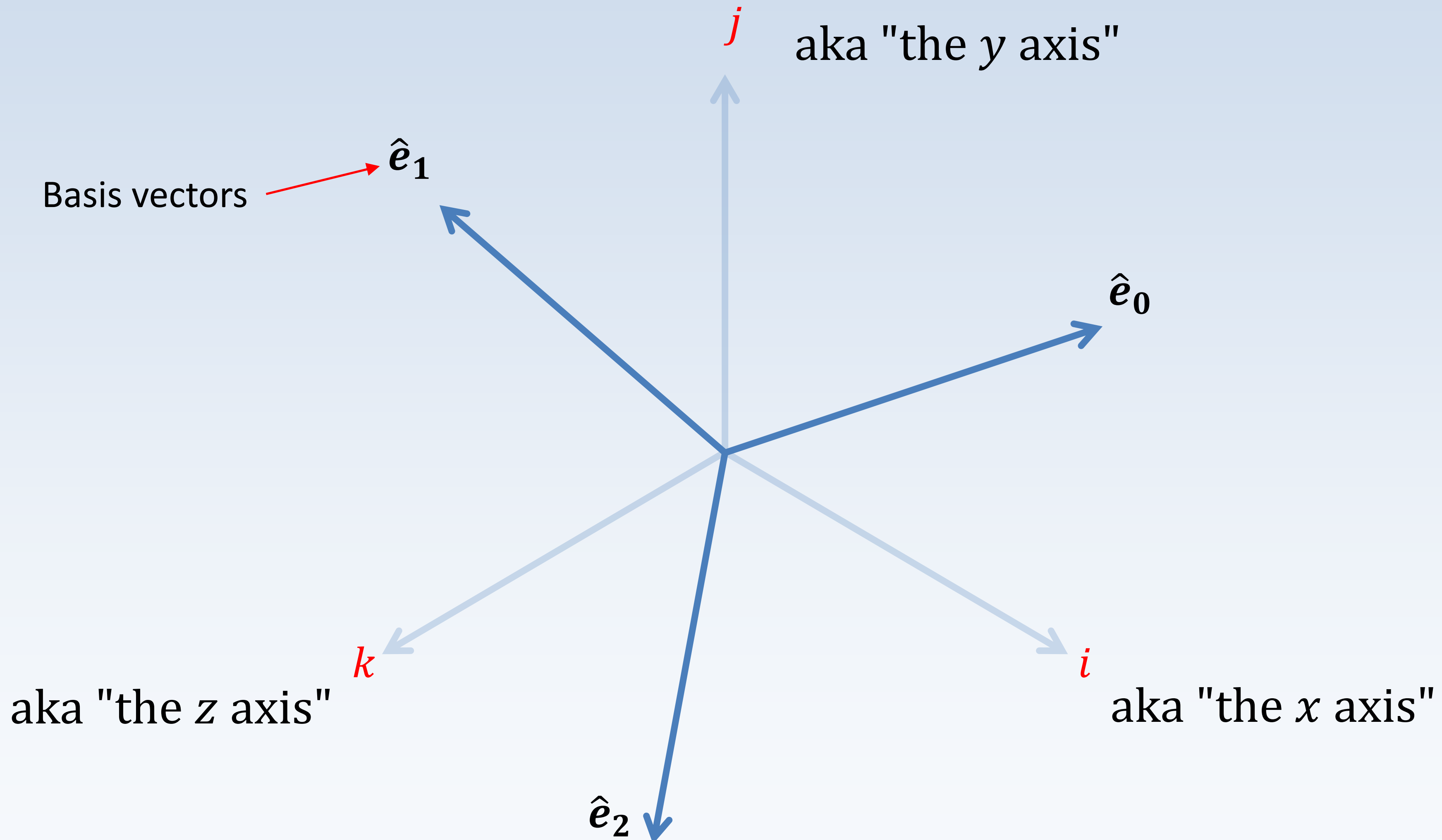
Rotation Matrices

- A 3x3 rotation matrix can be written as three normalized column vectors instead of nine elements:

$$R = [\hat{e}_0 \quad \hat{e}_1 \quad \hat{e}_2]$$

- These are the ***basis vectors*** for a coordinate system!

Rotation Matrices

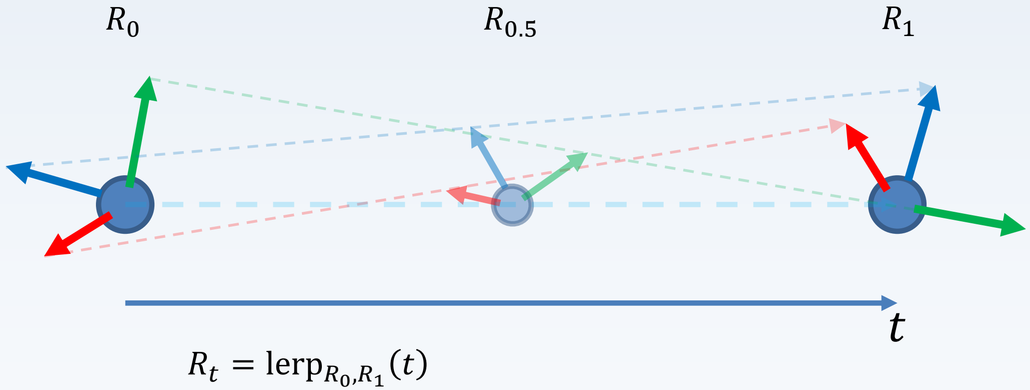


Rotation Matrices

- This is generally a tough concept to grasp
- One helpful way to understand basis vectors is to remove the notion of *absolute direction*
- Everything in animation/physics is *relative*
- Think of basis vectors as the *directions* ***relative*** to their parent coordinate frame!

Rotation Matrices

- Linearly interpolating rotation matrices:
- This has the same effect as applying *scale* simultaneously…
- Here's a graphical example (over time):



Rotation Matrices

- Rotation matrices are usually constructed from Euler angles (concatenate 3 matrices)
- Three separate rotations, one for each axis, multiplied together to give us one rotation
- *HUGE* problem with this…???
- ***Gimbal lock***

https://en.wikipedia.org/wiki/Gimbal_lock

Rotation Matrices

- Matrices are *required* for rendering…
- …but they are terrible for animation/physics…
- Cannot LERP matrices without consequences

- If only there was a ***<u>tool</u>*** to alleviate gimbal lock and *<u>animate rotations</u>* without the headache…

Quaternions

- First described by *William Rowan Hamilton*, an Irish mathematician, in 1843 (published 1865)
- "Vectors are 3D therefore they should represent rotations in 3D… right?"
- No matter how hard he tried, could not figure out how this worked…
- …until one day, while walking across the Brougham Bridge in Ireland…

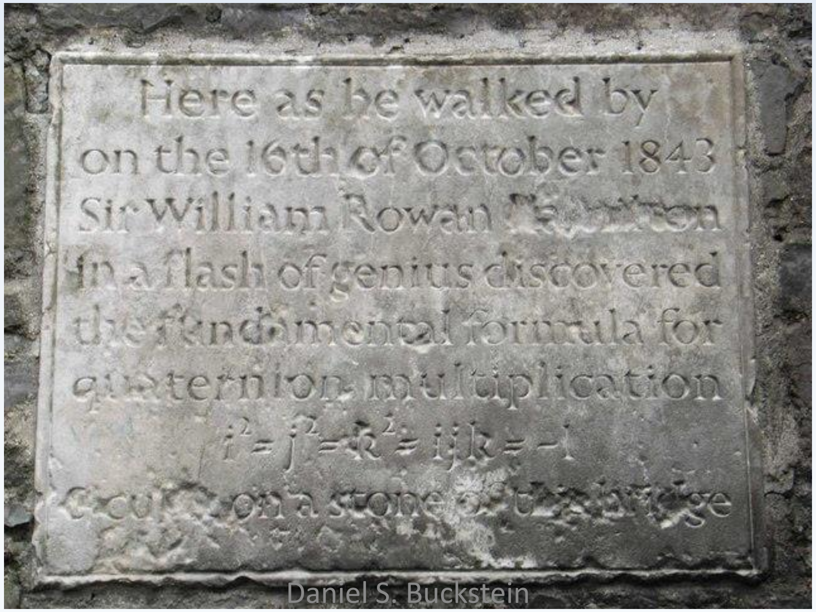
Quaternions

- …Hamilton figured out that a *fourth* component, a *real number*, would be required to control the rotation!
- Four parts → Quaternary
- The concept that frightens many
- …but actually not that scary
- Learning what they are and what they can do will save you in animation/physics

Quaternions

- Hamilton discovered that the basis elements are related by this identity:

$$\boldsymbol{i^2 = j^2 = k^2 = ijk = -1}$$

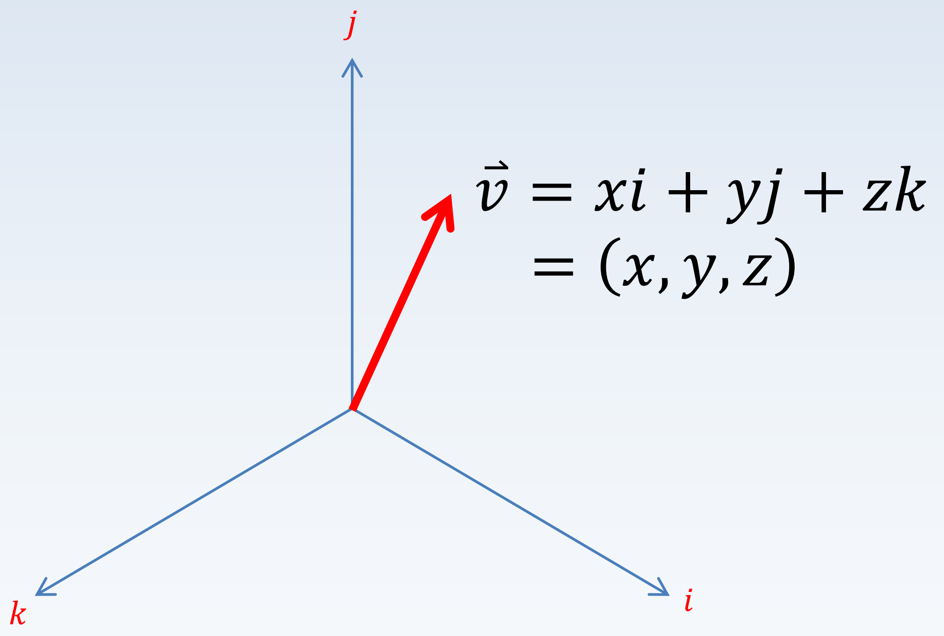


Quaternions

- Vectors are not real numbers, but are rather the sum of three imaginary components:

$$\vec{v} = xi + yj + zk$$

where *x, y,* and *z* are scalars along the respective *basis elements i, j* and *k*



Quaternions

- Adding a fourth basis element, a *real number*, gives us a quaternion:

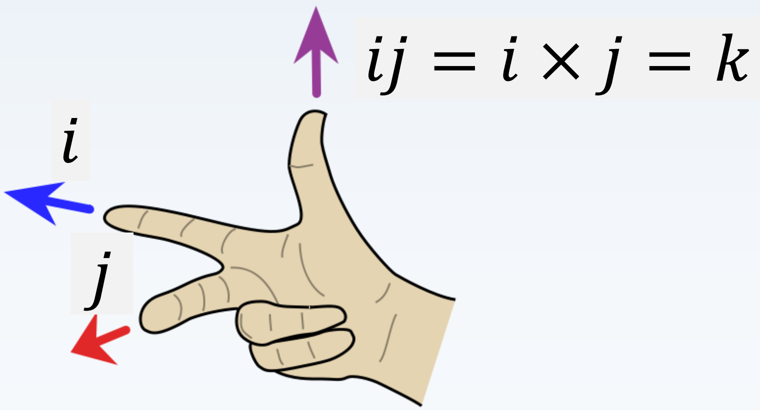
$$q = w(1) + xi + yj + zk$$

where 1, *i*, *j*, and *k* are the basis elements and *w*, *x*, *y*, and *z* are the respective scalars

Quaternions

- The basis elements *i, j* and *k* are related to each other, too:

$$\boldsymbol{ij} = \boldsymbol{k}, \qquad \boldsymbol{ji} = -\boldsymbol{k}$$
$$\boldsymbol{jk} = \boldsymbol{i}, \qquad \boldsymbol{kj} = -\boldsymbol{i}$$
$$\boldsymbol{ki} = \boldsymbol{j}, \qquad \boldsymbol{ik} = -\boldsymbol{j}$$



Quaternions

- Expressed mathematically, a quaternion is the *sum of a scalar and a vector*:

$$q = w + xi + yj + zk$$
$$\vec{v} = 0 + xi + yj + zk$$

Therefore,

$$q = w + \vec{v} = (w, \vec{v}) = (w, x, y, z)$$

Quaternions

- A quaternion with no real part is just a vector:

$$q = 0 + xi + yj + zk = (0, x, y, z) = (0, \vec{v})$$

- This is called a "*pure quaternion*"

- …it's just a vector.

Quaternions

- For all intents and purposes, quaternions share the same main functionalities of vectors… but in 4 dimensions:
- Dot product: $q_0 \cdot q_1 = w_0 w_1 + x_0 x_1 + y_0 y_1 + z_0 z_1$
- Magnitude: $\|q\| = \sqrt{w^2 + x^2 + y^2 + z^2}$, $\|q\|^2 = q \cdot q$
- Normalize: $\hat{q} = \frac{q}{\|q\|}$

Quaternions

- We are concerned with quaternions that have a length of *one* (normalized)
- Normalized quaternions represent *rotations*!
  - Called "versors" in pure math terms
- Rotation quaternions have huge advantages:
- Directly translates to axis-angle form
- No Euler angles, no gimbal lock
- Each quaternion maps to exactly *one* rotation!

Quaternions

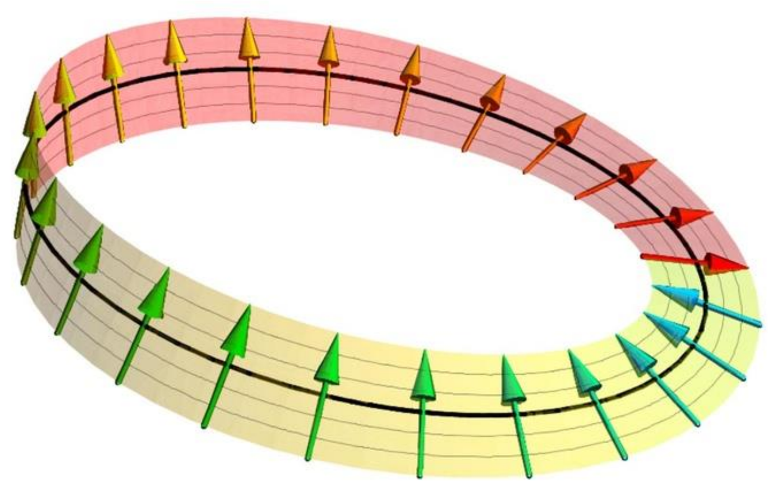
- ***Converting axis-angle to quaternion***:
- Given some angle θ and some normalized axis $\hat{n}$, a rotation quaternion is synthesized as:

$$w = \cos\left(\frac{\theta}{2}\right), \qquad \vec{v} = \hat{n}\sin\left(\frac{\theta}{2}\right)$$

$$\hat{q} = (w, \vec{v}) = \left(\cos\left(\frac{\theta}{2}\right), \hat{n}\sin\left(\frac{\theta}{2}\right)\right)$$

Quaternions

- Wait… why the *half*-angle???
- The behavior of a quaternion rotation forms a *spinor*, or *Mobius strip*:
- A full rotation is 720° for a spinor
- Half angle is the 3D equivalent



Quaternions

- The identity quaternion (no rotation) is

$$\hat{q} = \left(1, \vec{0}\right) = (1, 0, 0, 0)$$

- This is what you get from a $0°$ rotation, regardless of the axis:

$$\hat{q} = \left(\cos\left(\frac{0}{2}\right), \hat{n}\sin\left(\frac{0}{2}\right)\right) = (\cos 0\,, \hat{n}\sin 0)$$

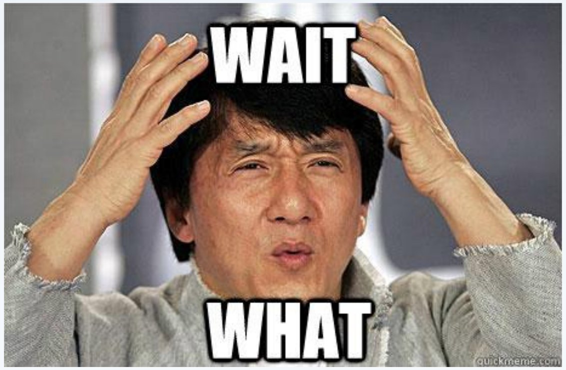
$$= (1, 0\hat{n}) = (1, 0, 0, 0)$$

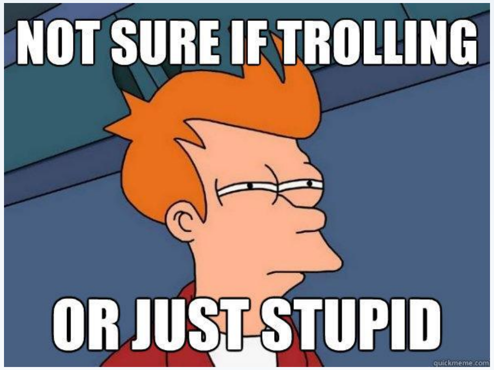
Quaternions

- What about a $360°$ $(2\pi$ rad) rotation about any axis? Should be the same, right???

$$\hat{q} = \left(\cos\left(\frac{2\pi}{2}\right), \hat{n}\sin\left(\frac{2\pi}{2}\right)\right) = (\cos\pi, \hat{n}\sin\pi)$$

$$= ({\color{red}-1}, 0\hat{n}) = ({\color{red}-1}, 0, 0, 0)$$





Quaternions

- In theory, a $360°$ rotation is the same as a $0°$ rotation… but because of the *spinor* shape…
- …it would take a full $720°$ rotation to return to the "original" orientation
- Rotation quaternions have a special property:

$$\hat{\boldsymbol{q}} \equiv -\hat{\boldsymbol{q}}$$

which means that a quaternion *and its negative* have the exact same *meaning*!

(Triple-bar equals sign is ***logical equivalence***: not *equal*, but do the same thing)

Quaternions

- So if the negative rotation quaternion represents the exact same rotation…
- …how do we determine the *inverse*???
- ***Conjugate***:

For any quaternion

$$q = (w, \vec{v}) = (w, x, y, z)$$

the *conjugate* is

$$q^* = (w, -\vec{\boldsymbol{v}}) = (w, -\boldsymbol{x}, -\boldsymbol{y}, -\boldsymbol{z})$$

Quaternions

- For any quaternion $q = (w, \vec{v})$, the inverse is

$$q^{-1} = \frac{q^*}{\|q\|^2}$$

which means that for a rotation quaternion (which has a magnitude of 1), the inverse is

$$\hat{q}^{-1} = \hat{q}^*$$

…just like how a rotation matrix's inverse is just the transpose!!!

Quaternions

- Why is the ***conjugate*** the inverse and not the ***negative***???
- We already saw that the *negative* quaternion represents the same rotation…
- If we flip the *axis* while using the same *angle*, the result is the opposite rotation
- Negating the entire quaternion is the same as flipping both the axis *and* the angle (because cosine!)

Quaternions

- How do we extract an axis and an angle from a quaternion???

$$w = \cos\left(\frac{\theta}{2}\right) \quad \rightarrow \quad \theta = 2\cos^{-1}(w)$$

$$\vec{v} = \hat{n}\sin\left(\frac{\theta}{2}\right) \quad \rightarrow \quad \hat{n} = \frac{1}{\sin\left(\frac{\theta}{2}\right)}\vec{v} \text{ or } \hat{n} = \frac{\vec{v}}{|\vec{v}|}$$

Quaternions

- **Concatenation (multiplication)**:
- The long way: take the product of two mathematical quaternions

$$q_0 = (w_0, \vec{v}_0) = (w_0, x_0, y_0, z_0)$$
$$= w_0 + x_0 i + y_0 j + z_0 k$$

$$q_1 = (w_1, \vec{v}_1) = (w_1, x_1, y_1, z_1)$$
$$= w_1 + x_1 i + y_1 j + z_1 k$$

Quaternions

- **Concatenation (full expansion)**:

$$q_0 q_1 = (w_0 + x_0 i + y_0 j + z_0 k)(w_1 + x_1 i + y_1 j + z_1 k)$$

$$\begin{aligned} &= w_0 w_1 + w_0 x_1 i + w_0 y_1 j + w_0 z_1 k \\ &\quad + x_0 w_1 i + x_0 i x_1 i + x_0 i y_1 j + x_0 i z_1 k \\ &\quad + y_0 w_1 j + y_0 j x_1 i + y_0 j y_1 j + y_0 j z_1 k \\ &\quad + z_0 w_1 k + z_0 k x_1 i + z_0 k y_1 j + z_0 k z_1 k \end{aligned}$$

Quaternions

- **Concatenation**:

$$q_0 q_1 = (w_0 + x_0 i + y_0 j + z_0 k)(w_1 + x_1 i + y_1 j + z_1 k)$$

$$\begin{aligned} = & \, w_0 w_1 + w_0 x_1 i + w_0 y_1 j + w_0 z_1 k \\ & + x_0 w_1 i + x_0 x_1 i^2 + x_0 y_1 ij + x_0 z_1 ik \\ & + y_0 w_1 j + y_0 x_1 ji + y_0 y_1 j^2 + y_0 z_1 jk \\ & + z_0 w_1 k + z_0 x_1 ki + z_0 y_1 kj + z_0 z_1 k^2 \end{aligned}$$

$ij = k$, $ji = -k$

$jk = i$, $kj = -i$

$ki = j$, $ik = -j$

$i^2 = j^2 = k^2 = ijk = -1$

Quaternions

- **Concatenation**:

$$q_0 q_1 = (w_0 + x_0 i + y_0 j + z_0 k)(w_1 + x_1 i + y_1 j + z_1 k)$$

$$\begin{aligned} &= w_0 w_1 + w_0 x_1 i + w_0 y_1 j + w_0 z_1 k \\ &\quad + x_0 w_1 i - x_0 x_1 + x_0 y_1 k - x_0 z_1 j \\ &\quad + y_0 w_1 j - y_0 x_1 k - y_0 y_1 + y_0 z_1 i \\ &\quad + z_0 w_1 k + z_0 x_1 j - z_0 y_1 i - z_0 z_1 \end{aligned}$$

$$ij = k, \quad ji = -k \qquad i^2 = j^2 = k^2 = ijk = -1$$

$$jk = i, \quad kj = -i$$

$$ki = j, \quad ik = -j$$

Quaternions

- **Concatenation**:

$$q_0 q_1 = (w_0 + x_0 i + y_0 j + z_0 k)(w_1 + x_1 i + y_1 j + z_1 k)$$

$$\begin{aligned} &= (w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1)1 \\ &+ (w_0 x_1 + x_0 w_1 + y_0 z_1 - z_0 y_1) i \\ &+ (w_0 y_1 - x_0 z_1 + y_0 w_1 + z_0 x_1) j \\ &+ (w_0 z_1 + x_0 y_1 - y_0 x_1 + z_0 w_1) k \end{aligned}$$

Quaternions

- **Concatenation**:
- Luckily there is a compact formula… so don't panic about all that mess:

$$q_0 = (w_0, \vec{v}_0), \qquad q_1 = (w_1, \vec{v}_1)$$

$$\boldsymbol{q_0 q_1} = \begin{pmatrix} \boldsymbol{w_0 w_1 - \vec{v}_0 \cdot \vec{v}_1} \ , \\ \boldsymbol{w_0 \vec{v}_1 + w_1 \vec{v}_0 + \vec{v}_0 \times \vec{v}_1} \end{pmatrix} \begin{matrix} \leftarrow \text{real part} \\ \leftarrow \text{vector part} \end{matrix}$$

Quaternions

- **Concatenation**:
- Like with matrices, concatenation is ***non-commutative***: $q_0 q_1 \neq q_1 q_0$
- Like with matrices (again), concatenation is ***associative***: $(q_0 q_1) q_2 = q_0 (q_1 q_2)$
- Like with matrices (yet again), the order of operation is ***right to left***: e.g. with $q = q_0 q_1$, *q1* happens first!

Quaternions

- **Rotating a vector**:
- Applying a rotation to a single vector using a quaternion is not as simple as it is with rotations…

$$\vec{v}' = R\vec{v} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Quaternions

- **Rotating a vector**:
- 4-dimensional math wizardry must be used
- First represent the vector as a *pure quaternion*: $\vec{v} = (0, x, y, z)$
- Then plug it into this formula (the result will also be a pure quaternion):

$$\vec{v}' = \hat{q}\vec{v}\hat{q}^*$$

Quaternions

- **Rotating a vector**:
- As always, there is a better formula:
- Let $\vec{v}$ represent the vector we want to rotate, and let $\vec{r}$ represent the quaternion's vector component:

Quaternion $\hat{q} = (w, \vec{r})$ rotating vector $\vec{v}$

$$\boldsymbol{\vec{v}' = \vec{v} + 2\vec{r} \times (\vec{r} \times \vec{v} + w\vec{v})}$$

Quaternions vs. Matrices

- **Performance (speed)**:
- Storage requirements:

Rotation matrix: 9 floats

Quaternion: ***4 floats***

$$R = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$\boxed{q = (w, x, y, z)}$$

Quaternions vs. Matrices

- **Performance (speed)**:
- Concatenation (chaining operations):

  Rotation matrices (R_0R_1): 45

  Quaternions (q_0q_1): ***28***

$$R_0R_1 = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$\boxed{\begin{aligned} & q_0q_1 \\ & = \begin{pmatrix} w_0w_1 - \vec{v}_0 \cdot \vec{v}_1 \;, \\ w_0\vec{v}_1 + w_1\vec{v}_0 + \vec{v}_0 \times \vec{v}_1 \end{pmatrix} \end{aligned}}$$

Quaternions vs. Matrices

- **Performance (speed)**:
- Rotating a vector:

  Rotation matrices ($R\vec{v}$): 15

  Quaternions ($\hat{q}\vec{v}\hat{q}^*$): ***30*** or ***41***

$$R\vec{v} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\boxed{\vec{v}' = \vec{v} + 2\vec{r} \times (\vec{r} \times \vec{v} + w\vec{v})}$$

$$\boxed{\vec{v}' = \hat{q}\vec{v}\hat{q}^*}$$

Quaternions vs. Matrices

- **Performance (general)**:
- Gimbal lock:
- Only a problem with *Euler angles*
- Since one quaternion maps to one rotation…
- No more Euler angles…
- … *no more gimbal lock*!!!

almost rip Apollo 13

Gimbal lock explained:
https://www.youtube.com/watch?v=zc8b2Jo7mno
https://www.youtube.com/watch?v=OmCzZ-D8Wdk

Quaternions vs. Matrices

- **Precision (correctness)**:
- Animation algorithms:
- Cannot animate rotation matrices ☹
- They are also slower to concatenate ☹☹
- Much more efficient to use something we *can* animate (using SLERP)
- Fear not quaternions!!! They are awesome!

Quaternion SLERP vs. Matrix LERP visualized:
https://www.youtube.com/watch?v=uNHIPVOnt-Y

Quaternions vs. Matrices

- **Performance vs. Precision**:
- The ubiquitous computer science dilemma
- Matrices suck but they are used for things quaternions can't handle
- Quaternion ***SLERP*** can be replaced with ***NLERP*** to save some time
- Later we'll discuss how to drop *homogeneous transforms* for a quaternion-related topic ;)

Applications of Quaternions

- **Conversion to rotation matrix**:
- Why would we want to convert a rotation quaternion to a matrix???
- Quaternions are a good *animation tool*…
- …but they do not play nicely with the other children ☹
- GPU does not know quaternions; so we will eventually require matrices for rendering

Applications of Quaternions

- **Conversion to rotation matrix**:
- For the rotation quaternion
  $q = (w, \vec{v}) = (w, x, y, z)$,
  the corresponding rotation matrix is

$$\boldsymbol{R_q} = \begin{bmatrix} \boldsymbol{w^2 + x^2 - y^2 - z^2} & \boldsymbol{2(xy - wz)} & \boldsymbol{2(xz + wy)} \\ \boldsymbol{2(xy + wz)} & \boldsymbol{w^2 - x^2 + y^2 - z^2} & \boldsymbol{2(yz - wx)} \\ \boldsymbol{2(xz - wy)} & \boldsymbol{2(yz + wx)} & \boldsymbol{w^2 - x^2 - y^2 + z^2} \end{bmatrix}$$

Applications of Quaternions

- Pro tip: who says quaternions only represent rotations? ;)

$$\hat{q} = (w, x, y, z)$$ ← *Normalized* quaternions are rotations!

$$s\hat{q} = (sw, sx, sy, sz)$$ ← …what about *un-normalized* quaternions?

Expand this and see what you get… what does the result imply???

$$R_q = \begin{bmatrix} (sw)^2 + (sx)^2 - (sy)^2 - (sz)^2 & 2(sxsy - swsz) & 2(sxsz + swsy) \\ 2(sxsy + swsz) & (sw)^2 - (sx)^2 + (sy)^2 - (sz)^2 & 2(sysz - swsx) \\ 2(sxsz - swsy) & 2(sysz + swsx) & (sw)^2 - (sx)^2 - (sy)^2 + (sz)^2 \end{bmatrix}$$

Applications of Quaternions

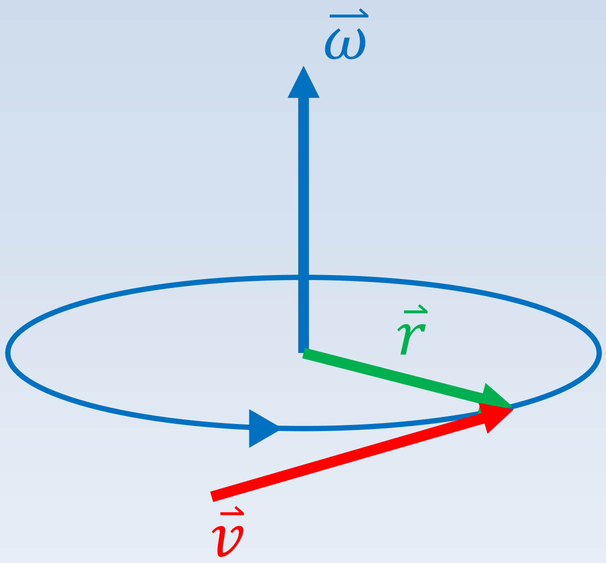
- Applications in physics:
- Derivative of a quaternion over time:
- If q_t is the rotation of an object at time t, then

$$\boldsymbol{\frac{d}{dt}q_t = \frac{1}{2}\omega_t q_t}$$

is the change in orientation over time, where ω is the angular velocity

Applications of Quaternions

- Applications in physics:
- Angular velocity as a vector:



$$\vec{\omega} = \frac{\vec{r} \times \vec{v}}{\|\vec{r}\|^2}$$

where

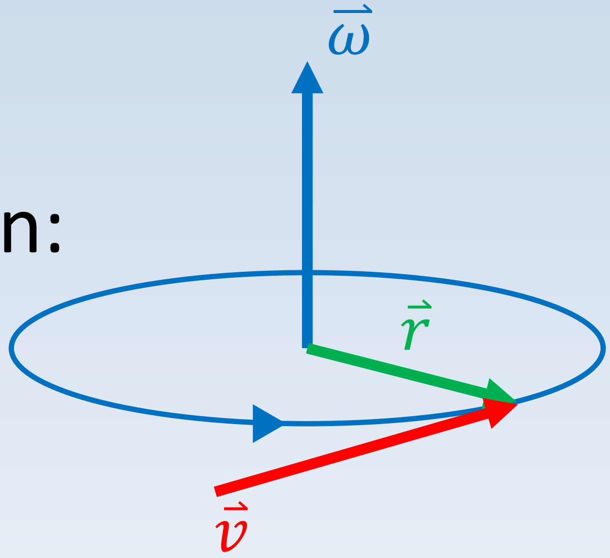
ω = angular velocity,

r = vector from center of mass (axis) to particle,

v = tangential velocity at particle

Applications of Quaternions

- Applications in physics:
- Angular velocity as a quaternion:



$$\omega_t = (0, x_\omega, y_\omega, z_\omega)$$

$$\frac{d}{dt} q_t = \frac{1}{2} \omega_t q_t$$

…and use that to integrate!

Applications of Quaternions

- Applications in physics:
- Integrating angular velocity into rotation:
- Compare with explicit Euler for position:

$$x_{t+dt} = x_t + \frac{dx}{dt}dt$$

$$x_{t+dt} = x_t + v_t dt$$

$$v_{t+dt} = v_t + a_t dt \qquad F_t = m\, a_t$$

Applications of Quaternions

- Applications in physics:
- Integrating angular velocity into rotation:
- Euler works for quaternions too:

$$q_{t+dt} = q_t + \frac{dq}{dt} dt$$

$$q_{t+dt} = q_t + \omega_t q_t \, dt/2$$

$$\omega_{t+dt} = \omega_t + \alpha_t dt \qquad \tau_t = I \, \alpha_t$$

Applications of Quaternions

- Applications in physics:
- Problem: integrated value will have *scale* (because of floating point error)
- Need to *normalize* result to remove unwanted scaling:

$$q'_{t+dt} = \mathrm{normalize}(q_{t+dt})$$

The end.

- Questions? Comments? Concerns?