Working With 3D Rotations

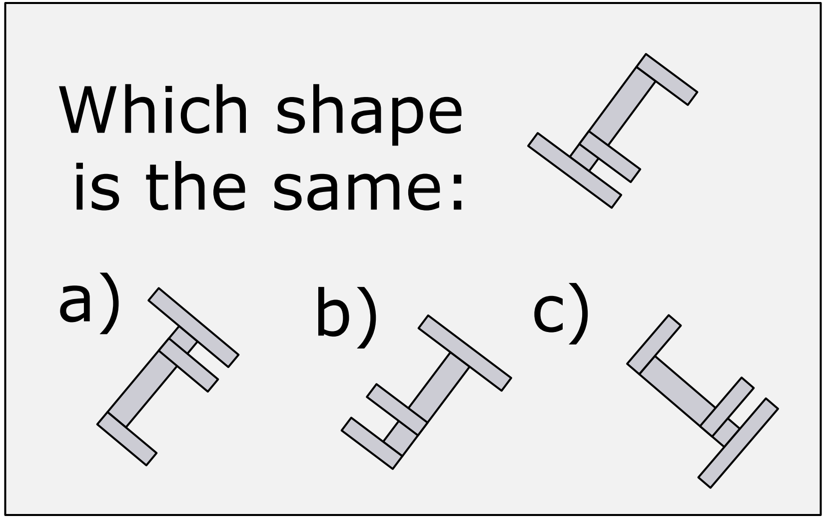
Stan Melax
Graphics Software Engineer, Intel

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21
2014

Human Brain is wired for Spatial Computation

"I don't need to ask for directions"

Translations



A childhood IQ test question

Rotations

Agenda

- Rotations and Matrices *(hopefully review)*
- Combining Rotations
- Matrix and Axis Angle
- Challenges of deep Space (of Rotations)
- Quaternions
- Applications

Terminology Clarification

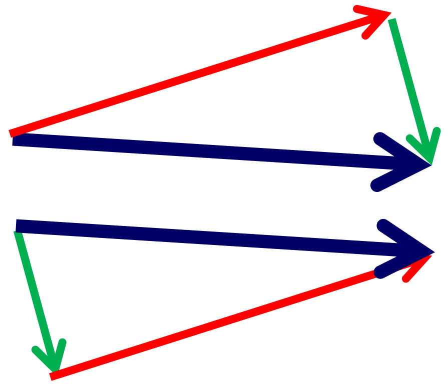
Preferred usages of various terms:

	Linear	Angular
Object Pose	Position (point)	Orientation
A change in Pose	Translation (vector)	Rotation
Rate of change	Linear Velocity	Spin

also: *Direction* specifies 2 DOF, *Orientation* specifies all 3 angular DOF.

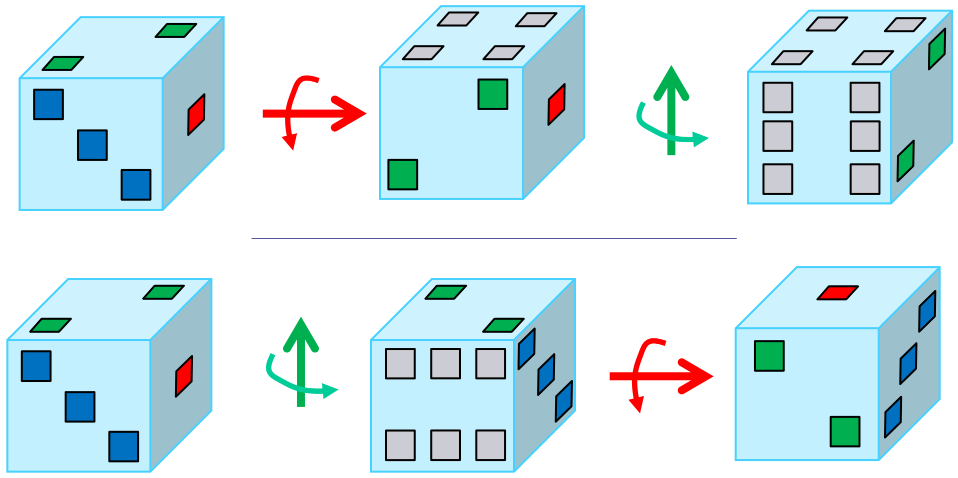
Rotations Trickier than Translations

Translations



a then **b** == **b** then **a**

Rotations

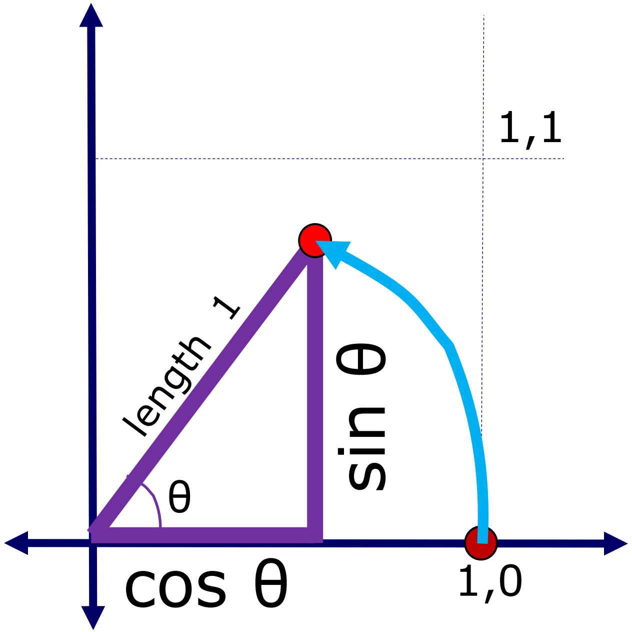


x then **y** != **y** then **x**

(non-commutative)

- Programming with rotations also more challenging!

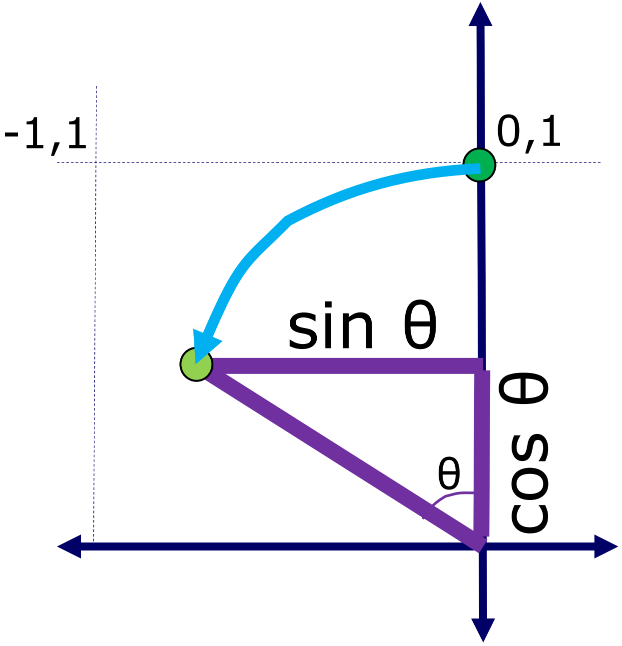
2D Rotation θ



Rotate ● [1 0] by θ about origin

● [cos(θ) sin(θ)]

2D Rotation θ

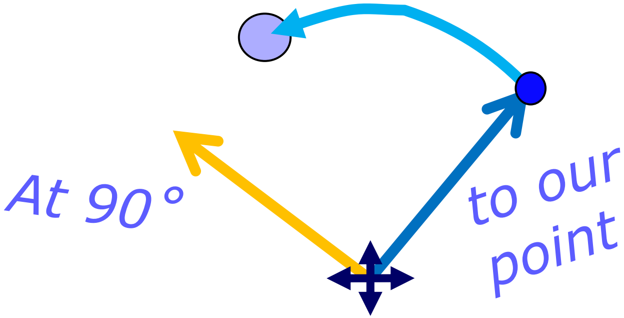


Rotate ● [0 1] by θ about origin

● [-sin(θ) cos(θ)]

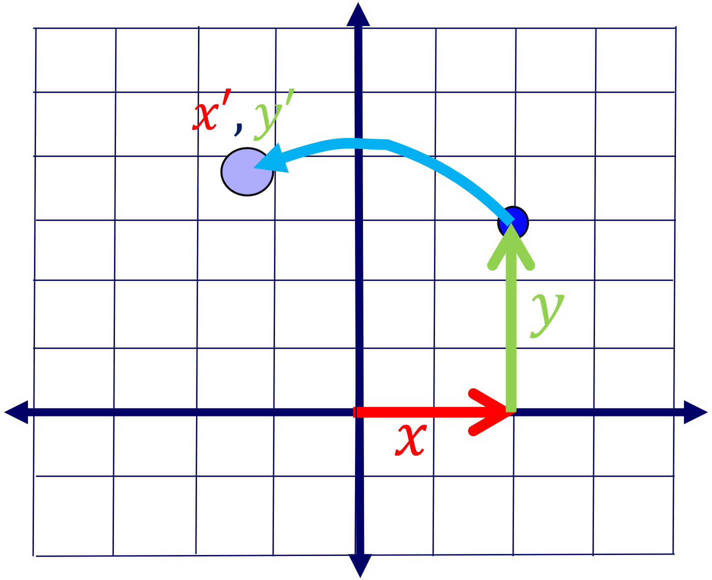
2D Rotation of an arbitrary point

Rotate about origin by θ



$= \cos\theta \quad + \sin\theta$

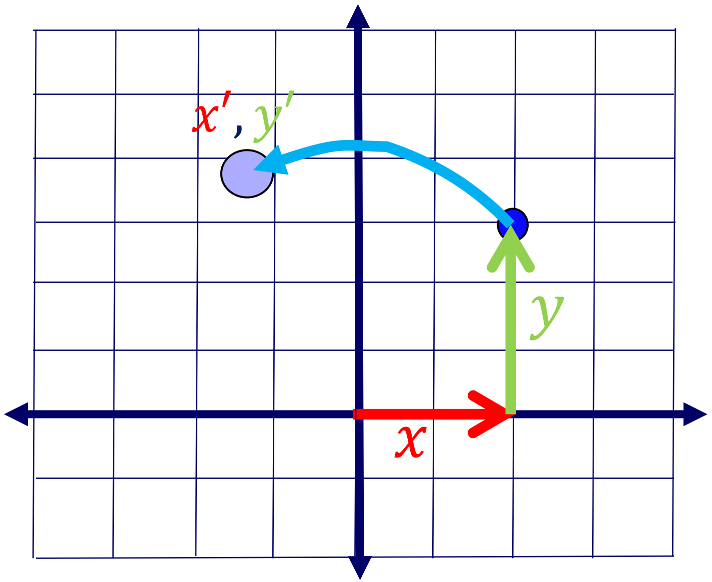
2D Rotation of an arbitrary point



Rotate $\begin{bmatrix} x \\ y \end{bmatrix}$ about origin by θ

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

2D Rotation Matrix



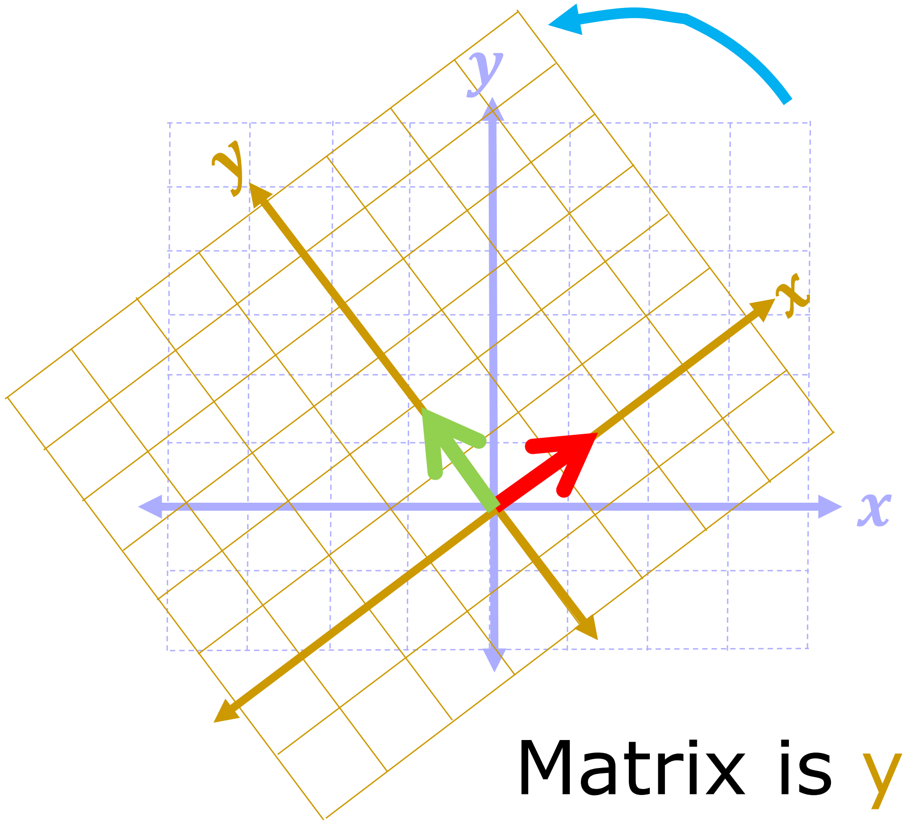
Rotate $\begin{bmatrix} x \\ y \end{bmatrix}$ about origin by θ

- $x' = x\cos\theta - y\sin\theta$
  $y' = x\sin\theta + y\cos\theta$

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

Matrix $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ is rotation by θ

2D Orientation



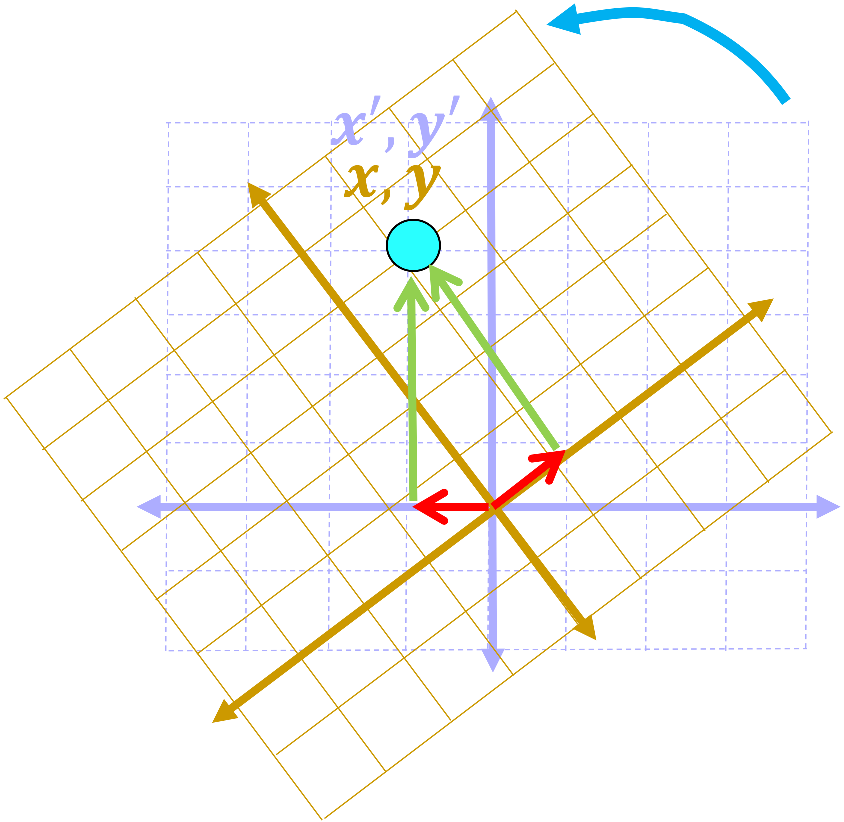
Yellow grid placed over first grid but at angle of θ

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Columns of the matrix are the directions of the axes.

Matrix is yellow grid's <u>Orientation</u>

2D Passive Transformation



Basis: $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

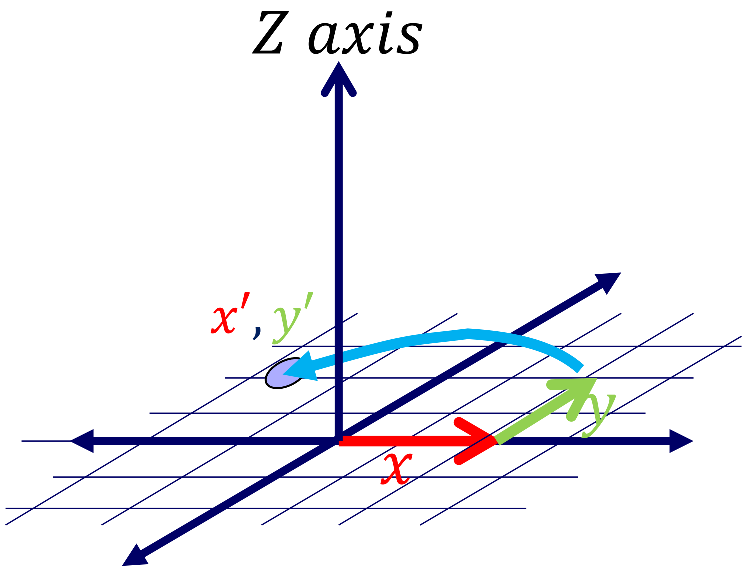
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

(note: exact same math as before)

$\begin{bmatrix} x' \\ y' \end{bmatrix}, \begin{bmatrix} x \\ y \end{bmatrix}$ both same point but

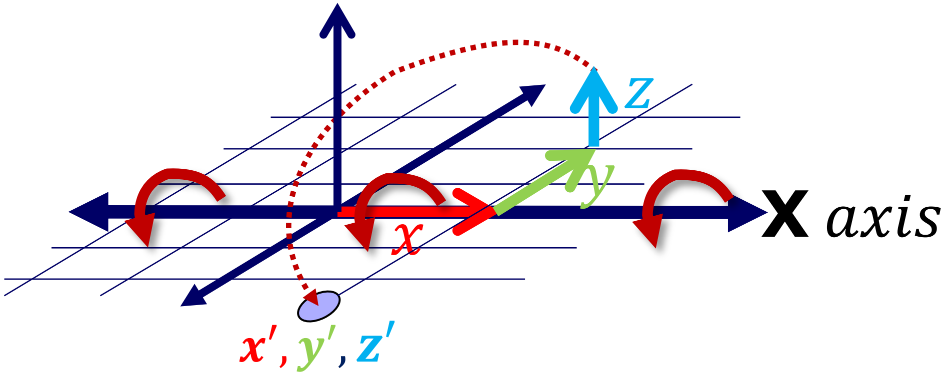
In different reference frames

3D Rotation around Z axis

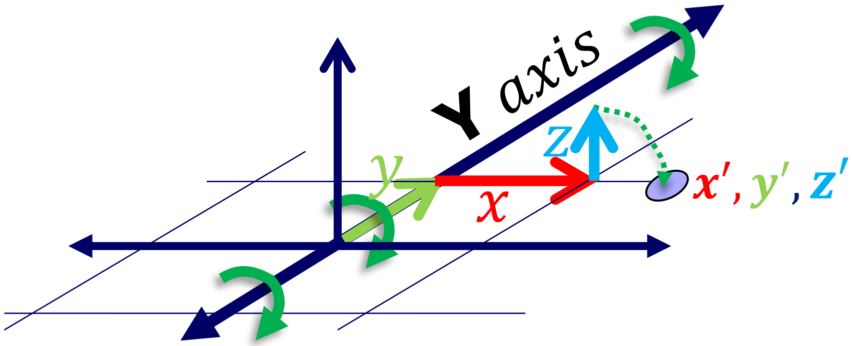


$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Can Rotate around X and Y too



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

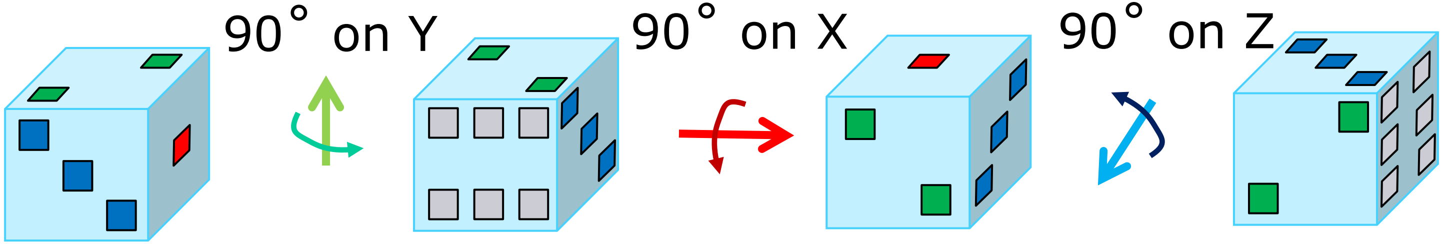


- $$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Rotating Objects (changing orientation)

Rotations:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Orientations:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Matrices used for both rotations and orientations

Quat rotations: [0 .7 0 .7] [.7 0 0 .7] [0 0 .7 .7]

orientations: [0 0 0 1] [0 0 .7 .7] [.5 .5 .5 .5] [0 .7 .7 0]

Row vs Column Conventions

OpenGL and most math books use column vectors:

$$\mathbf{v}' = \mathbf{M}\,\mathbf{v} = \mathbf{B}\,\mathbf{A}\,\mathbf{v}$$

Some engines, APIs (DirectX) use row convention:

$$\mathbf{v}' = \mathbf{v}\,\mathbf{M}^T = \mathbf{v}\,\mathbf{A}^T\,\mathbf{B}^T$$

All the same.

Combining Rotations

Combine a sequence of Rotations **A**,**B**,...

Rotate **v** by **A**, then **B**, then **C**...

= C (B (Av))

Mathematically we know

C (B (A v)) == (C B A) v

So with matrix-matrix multiplication let:

R = C B A

R is a single rotation that is the same as rotating by **A**, then by **B** then **C**.

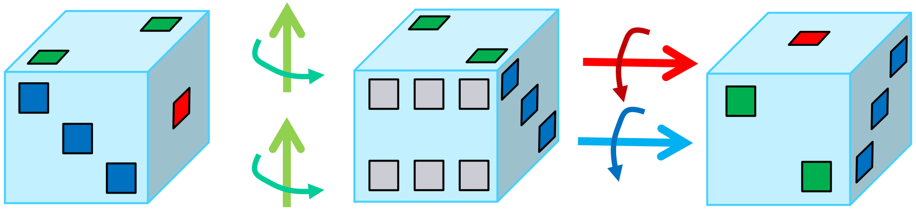
Multiplication Order:

WORLD

90° on World Y

90° on **"World" X**

$\boldsymbol{A}_{world}$ $\boldsymbol{B}_{world}$

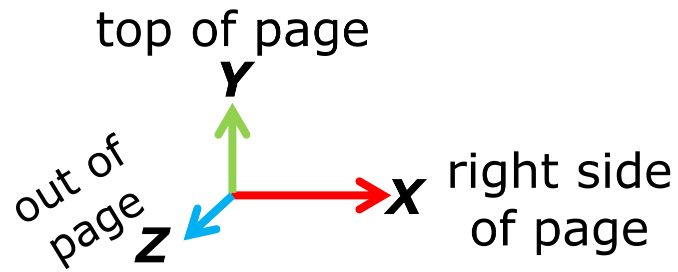


$\boldsymbol{A}_{local}$ $\boldsymbol{B}_{local}$

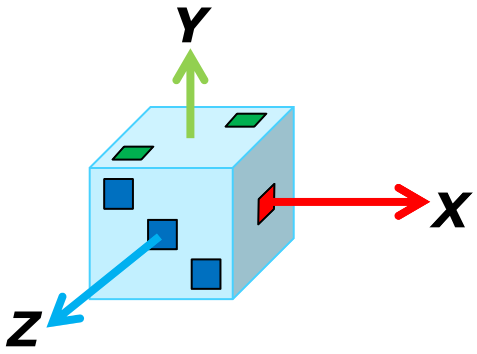
LOCAL

90° on Local Y (dice side 2)

90° on **"Local" Z** (dice side 3)



World Coordinate Frame



Dice Coordinate Frame

Multiplication Order: Math Equations

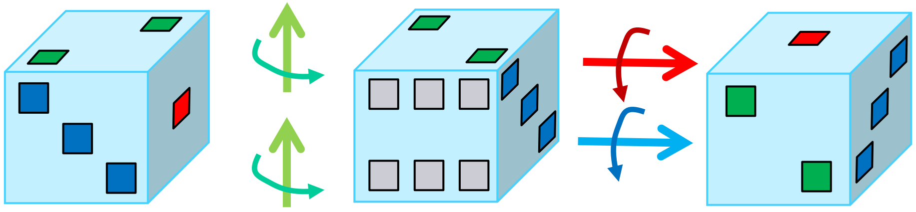
WORLD

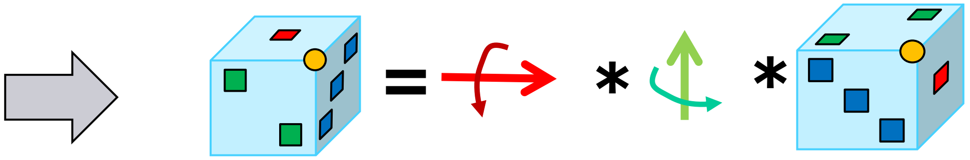
90° on World Y

90° on **"World" X**

A_{world}

B_{world}





$$dice_{new} = B_{world} * A_{world} * dice$$

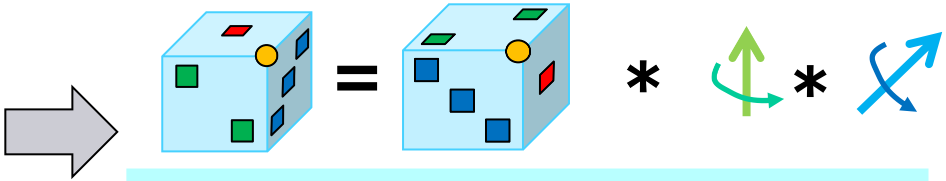
LOCAL

A_{local}

90° on Local Y (dice side 2)

B_{local}

90° on **"Local" Z** (dice side 3)

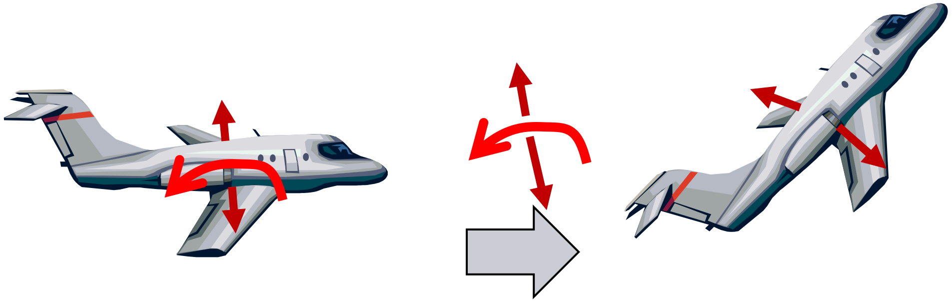


$$dice_{new} = dice * A_{local} * B_{local}$$

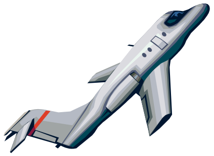
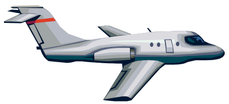
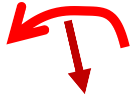
Both produce: 120° *on* [1 1 1]

Example When to use Local frame

- Player "pulls up" on flight stick.
- Pitch upward about <u>object</u> wing (**x**) axis.
- World **x** irrelevant
- Multiply rotation (about **x**) on the <u>right</u> hand side



Math:

 =  * $\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$ 

$$\mathit{climbing\ orientation} = \mathit{cruiseing\ orientation} * \mathit{pitch\_up\ rotation}$$

Sidenote: a point doesn't have an orientation, so never do this for points.

Find Rotation *R* Between Orientations *A* and *B*

need to be more specific

- Have an object with orientation ***A***, what rotation ***R*** will change it to have orientation ***B***?

$$\boldsymbol{R = BA^{-1}}$$

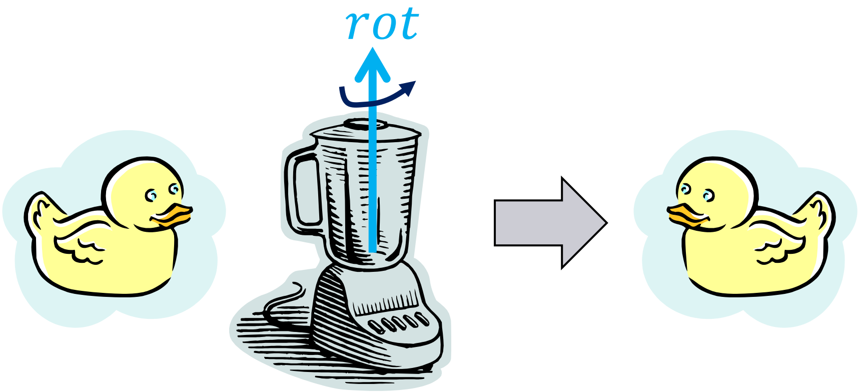
- Given a direction *v* in reference frame ***A***, what rotation ***R*** will show how *v* points according to ***B***?

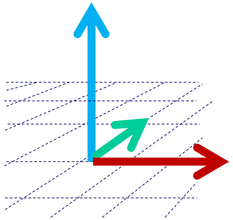
$$\boldsymbol{R = B^{-1}A}$$

Be aware of all the details of the problem to be solved.

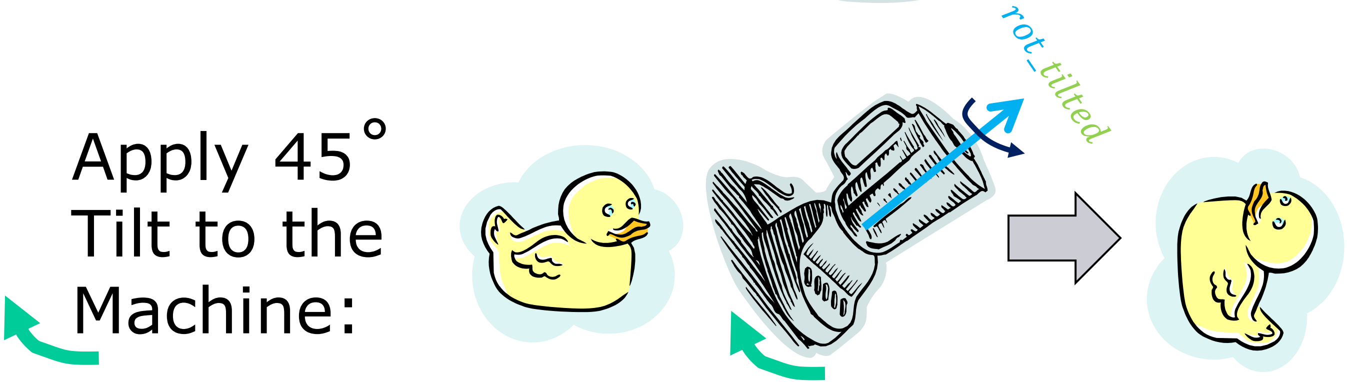
Rotating (Reorienting) a Rotation

Machine that rotates an object by rot :

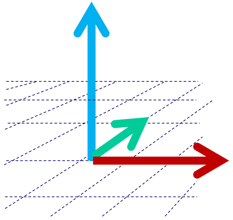


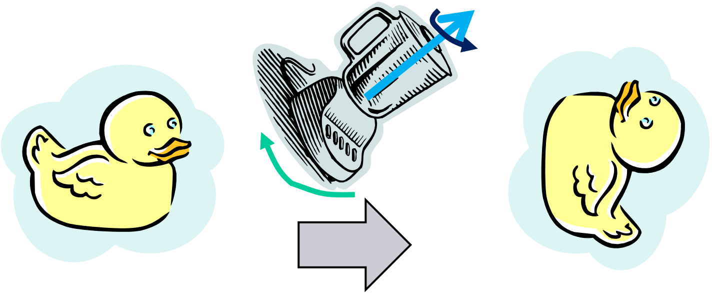


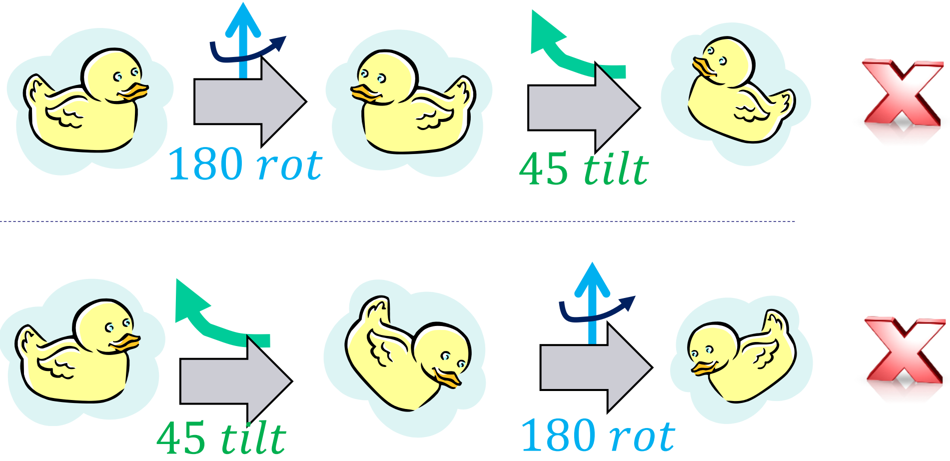
Apply 45° Tilt to the Machine:



Rotating a Rotation – *Its Different*



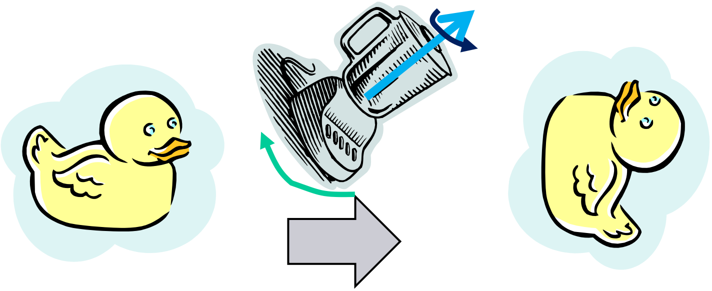




Neither of these multiplication sequences work
☹

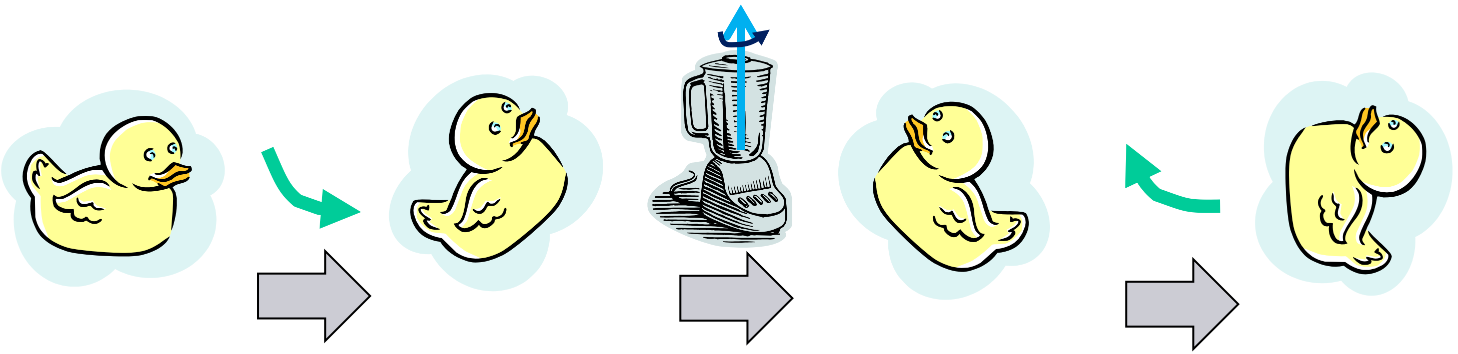
Rotating a Rotation: Decompose Steps

Tilted Machine:



How to calculate what this new rotation will be?

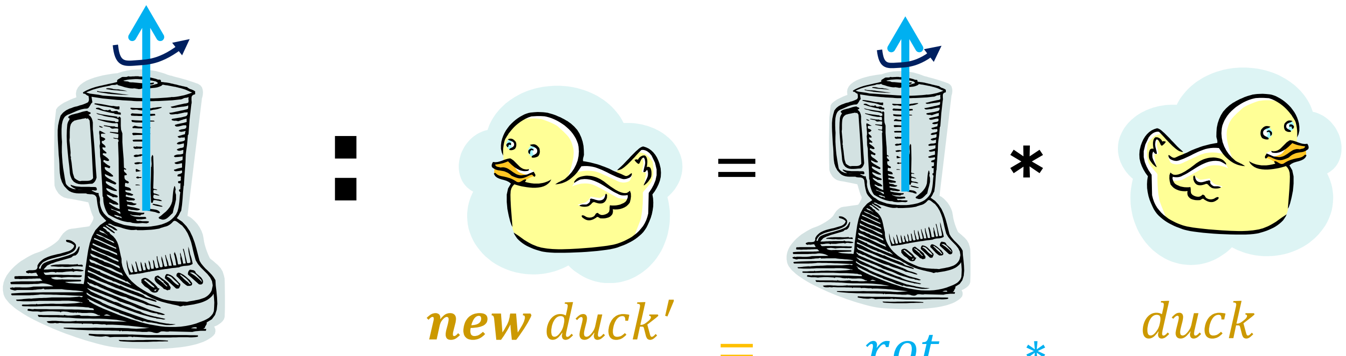
Rotate duck into **and** back out of the machine's reference frame:



Same Result!

Rotating a Rotation: The Mathematics

UPRIGHT



Initial equation

new duck′ orientation = *rot* * *duck orientation*

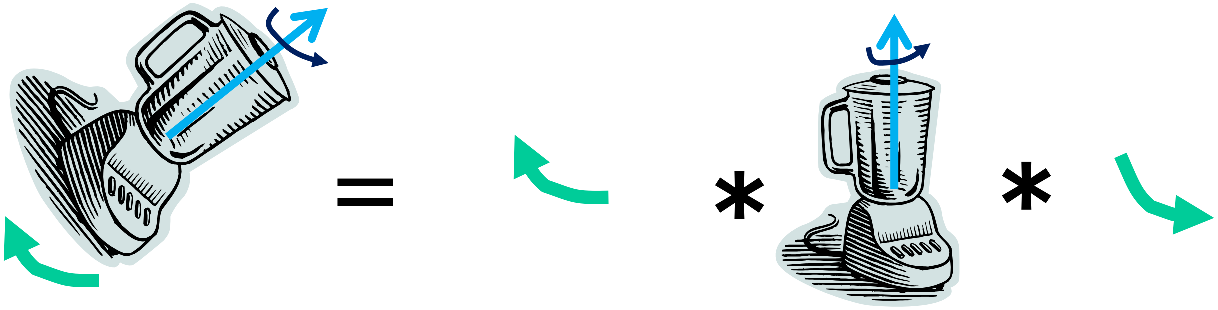
TILTED

equation w tilt

new duck″ = *tilt* * *rot* * $tilt^{-1}$ * *duck*

Rotating a Rotation: The Mathematics

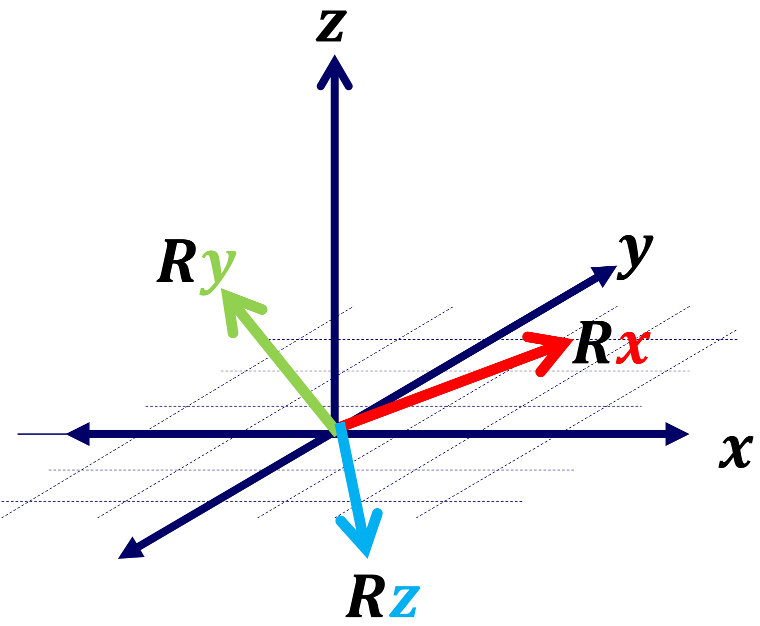
Now drop the duck…



$$rot_{tilted} = tilt * rot * tilt^{-1}$$

Matrix & Axis Angle

3D Orientation / Rotation Matrix *R*

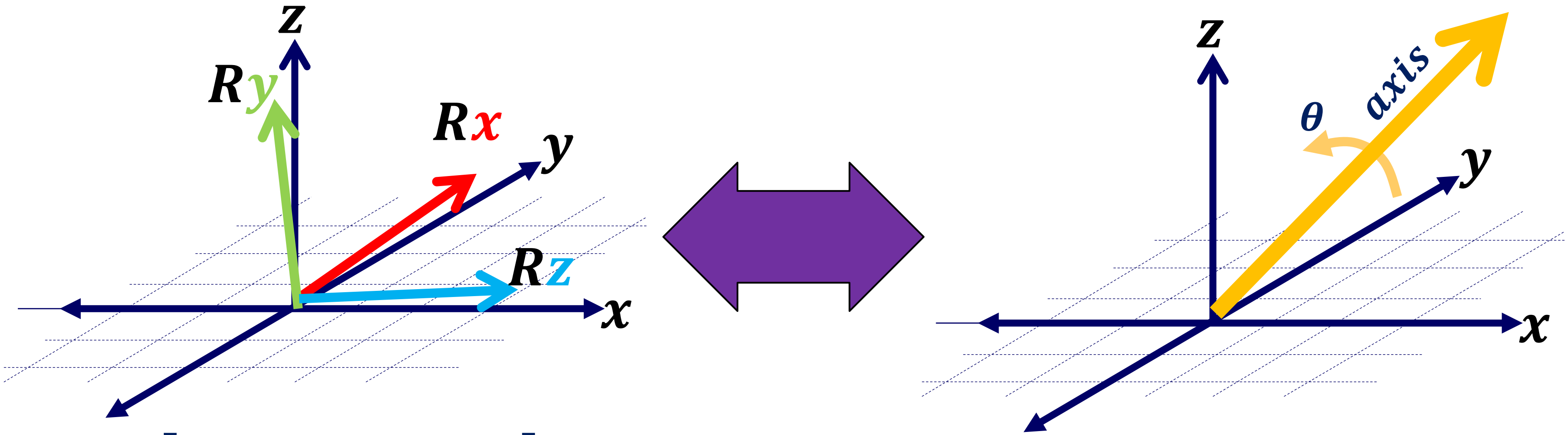


$$\boldsymbol{R} = \begin{bmatrix} Rx_x & Ry_x & Rz_x \\ Rx_y & Ry_y & Rz_y \\ Rx_z & Ry_z & Rz_z \end{bmatrix}$$

General form of Rotation Matrix:

- Orthonormal basis: $\boldsymbol{Rx}$ $\boldsymbol{Ry}$ $\boldsymbol{Rz}$
- $\boldsymbol{Rz} = \boldsymbol{Rx} \times \boldsymbol{Ry}$ etc.
- Determinant(R)==1
- Inverse(R) == Transpose(R)
- Has a corresponding axis of rotation

Rotation Matrix – Finding its Axis Angle

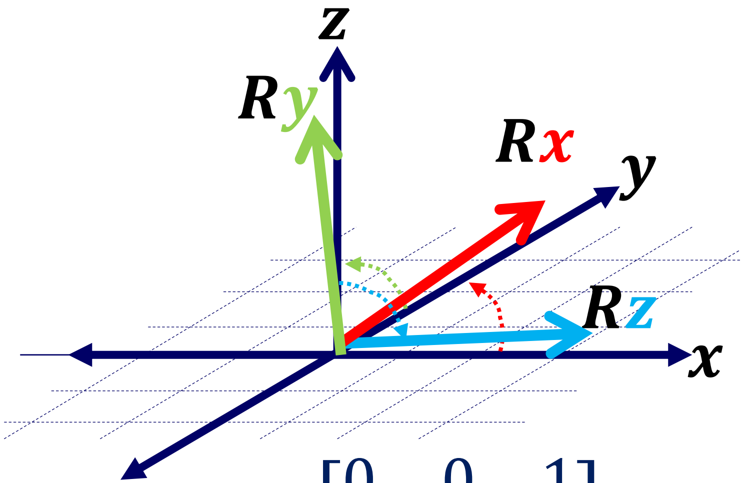


$$R = \begin{bmatrix} Rx_x & Ry_x & Rz_x \\ Rx_y & Ry_y & Rz_y \\ Rx_z & Ry_z & Rz_z \end{bmatrix}$$

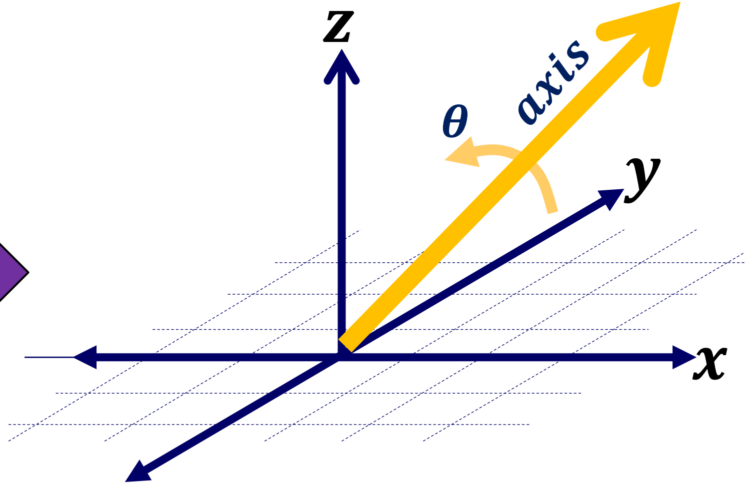
$$\boldsymbol{axis}, \boldsymbol{\theta}$$

$axis$ will be an eigenvector of $\boldsymbol{R}$

Example of corresponding Matrix and Axis Angle



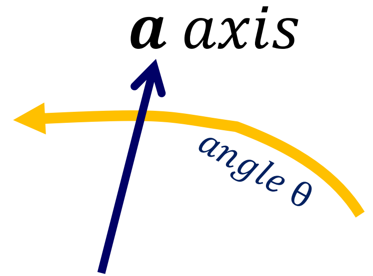
$$R = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



$$\boldsymbol{axis}, \boldsymbol{\theta} = [1\ 1\ 1], 120°$$

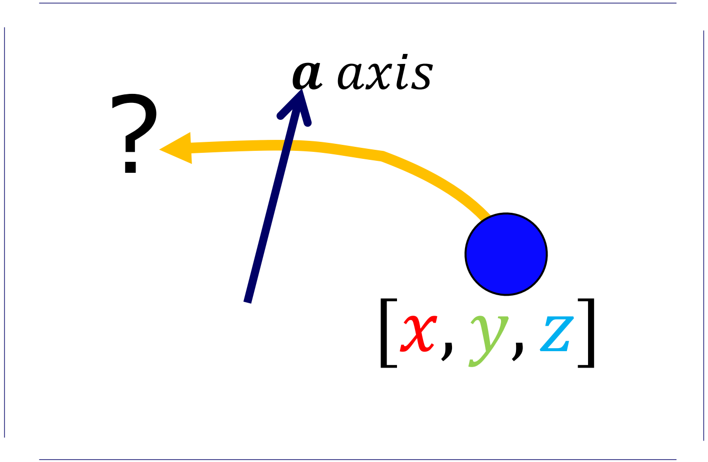
To check, verify: $\boldsymbol{axis} == \boldsymbol{R} * \boldsymbol{axis}$

Matrix from general axis ***a***, angle θ



*Matrix for **a**,θ ?*

Matrix from general axis ***a***, angle θ

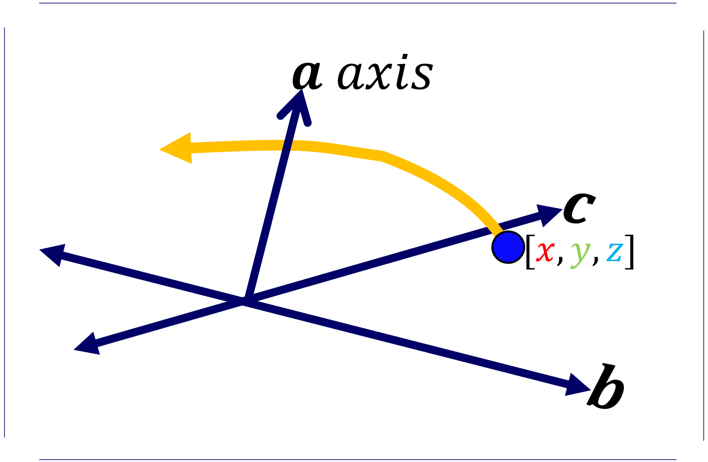


How would axis/angle rotate a point $[x, y, z]$*?*

Matrix from general axis *a*, angle θ

- Find ***b***,***c*** unit vecs ***a***,***b***,***c*** orthonormal

$\boldsymbol{a} = \boldsymbol{b} \times \boldsymbol{c}\,, \qquad \boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b}, \qquad \boldsymbol{b} = \boldsymbol{c} \times \boldsymbol{a}$

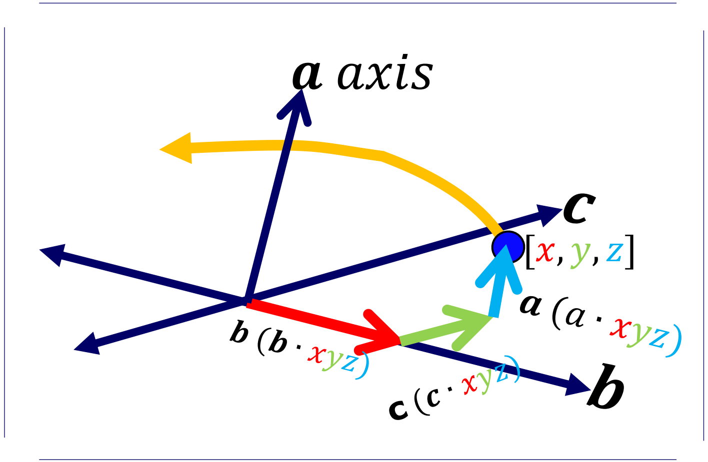


Matrix from general axis ***a***, angle θ

- Find ***b***,***c*** unit vecs ***a***,***b***,***c*** orthonormal

  $\boldsymbol{a} = \boldsymbol{b} \times \boldsymbol{c}\,, \qquad \boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b}, \qquad \boldsymbol{b} = \boldsymbol{c} \times \boldsymbol{a}$

- Get [xyz] as weighted sum of **a**,**b**,**c**

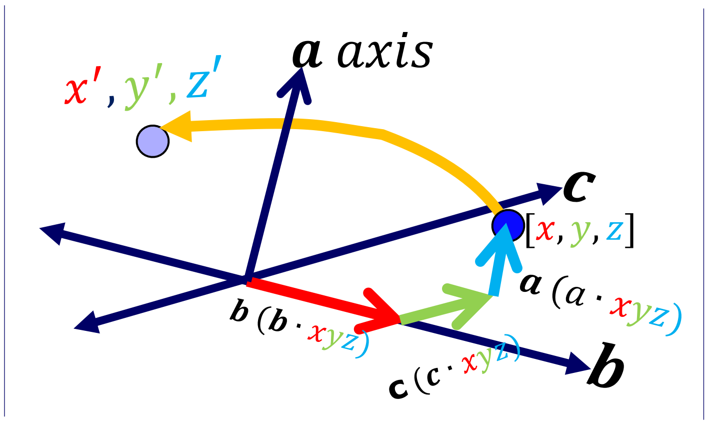


Matrix from general axis ***a***, angle θ

- Find ***b***,***c*** unit vecs ***a***,***b***,***c*** orthonormal

  $\boldsymbol{a} = \boldsymbol{b} \times \boldsymbol{c}\,, \qquad \boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b}, \qquad \boldsymbol{b} = \boldsymbol{c} \times \boldsymbol{a}$

- Get [xyz] as weighted sum of **a**,**b**,**c**
- Stuff along **a** stays the same,
- Results along **b** & **c** based on sinθ and cosθ portions along **b** & **c**



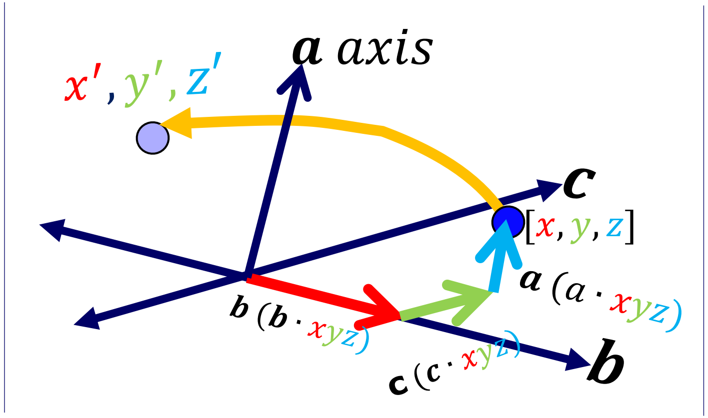
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \boldsymbol{a}\left(\boldsymbol{a} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) + \boldsymbol{b}\left(\boldsymbol{b} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \cos\theta - \boldsymbol{c} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \sin\theta\right) + \boldsymbol{c}\left(\boldsymbol{b} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \sin\theta + \boldsymbol{c} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \cos\theta\right)$$

Matrix from general axis ***a***, angle θ

- Find ***b***,***c*** unit vecs ***a***,***b***,***c*** orthonormal

  $\boldsymbol{a} = \boldsymbol{b} \times \boldsymbol{c}\,, \qquad \boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b}, \qquad \boldsymbol{b} = \boldsymbol{c} \times \boldsymbol{a}$

- Get [xyz] as weighted sum of **a**,**b**,**c**
- Stuff along **a** stays the same,
- Results along **b** & **c** based on sinθ and cosθ portions along **b** & **c**



$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \boldsymbol{a}\left(\boldsymbol{a} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) + \boldsymbol{b}\left(\boldsymbol{b} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \cos\theta - \boldsymbol{c} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \sin\theta\right) + \boldsymbol{c}\left(\boldsymbol{b} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \sin\theta + \boldsymbol{c} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \cos\theta\right)$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = (\boldsymbol{a}\boldsymbol{a}^T + \boldsymbol{b}\boldsymbol{b}^T \cos\theta - \boldsymbol{b}\boldsymbol{c}^T \sin\theta + \boldsymbol{c}\boldsymbol{b}^T \sin\theta + \boldsymbol{c}\boldsymbol{c}^T \cos\theta) \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

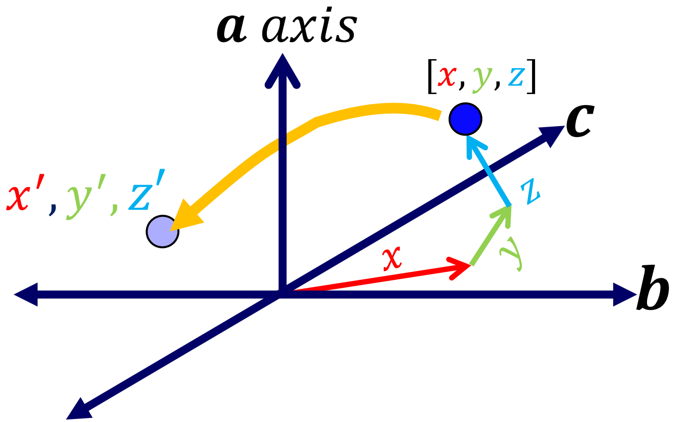
"still need method for finding b,c"

Matrix from general axis *a*, angle θ

Alternatively (Equivalently):

Think of [***b***,***c***,***a***] as a 3x3 basis.

- Move/rotate into **abc**’s reference frame.
- Do spin on ‘local’ **z** axis
- Rotate back out



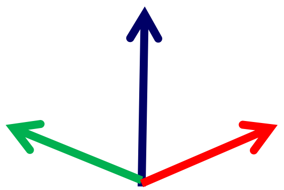
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = [\boldsymbol{b}\ \boldsymbol{c}\ \boldsymbol{a}] \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{c} \\ \boldsymbol{a} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$= \overbrace{\begin{bmatrix} b_x & c_x & a_x \\ b_y & c_y & a_y \\ b_z & c_z & a_z \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_x & b_y & b_z \\ c_x & c_y & c_z \\ a_x & a_y & a_z \end{bmatrix}}^{\text{3x3 Rotation Matrix}} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

“ok, but this math is still not concise.”

Challenges with the Space of Rotations

Matrix Disadvantages



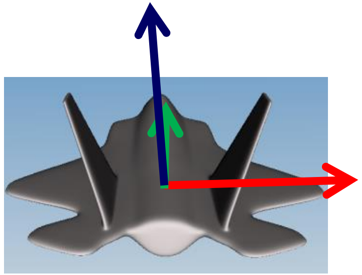
Great for some systems (batch rendering), but not ideal for animation, gameplay, or physics code.

$$\boldsymbol{R} = \begin{bmatrix} Rx_x & Ry_x & Rz_x \\ Rx_y & Ry_y & Rz_y \\ Rx_z & Ry_z & Rz_z \end{bmatrix}$$

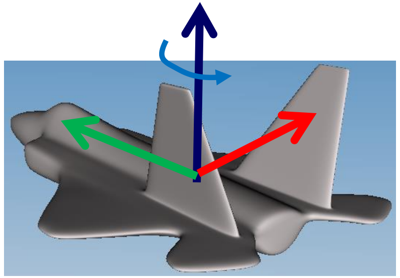
- Non-compact (9 floats for only 3DOF)
- Numerical Drift, non-orthonormal over time
- Getting meaningful information non-trivial?
  - Extracting an axis of rotation by eigenvector
- Interpolation between orientations (keyframes)

Is there a better way to be working with rotations/orientations?

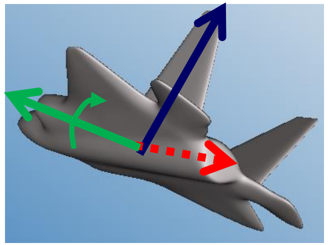
Yaw-Pitch-Roll (Euler angles)



y,p,r = 0,0,0



45,0,0

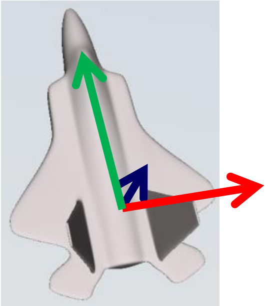


45,0,45

- Ordered sequence of rotations on 3 fixed main axes.
- Ideal representation for many game systems:
    - Standing NPC (yaw==heading)
    - Camera AI,
    - Helicopter flight.
- Convert to Matrix on the fly as necessary.

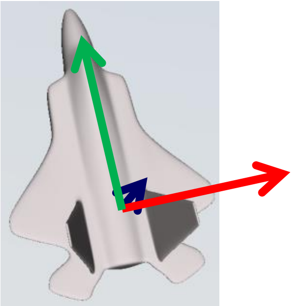
Yaw-Pitch-Roll – not ideal for general 3D

- Concatenating rotations: Done by matrix multiplication. Converting back to YPR? ☹
- Smooth interpolation and comparing rotations. What's the angle between:



[20 80 -20]

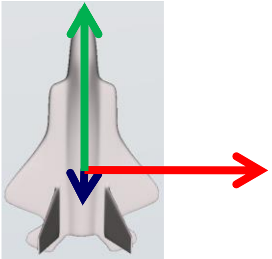
and



[-60 80 60]

Numerically distant, but Orientations similar!

Consider pitch upward to 90:



Could be:
[0 90 0] or
[45 90 -45] or
[n 90 –n] (any n)

Angle Axis

Axis Angle has Potential:

- General 3D
- Compact (drift averse)
- Inversion and Interpolation easy (just modify angle)

Issues:

- Specifics of the encoding (angle as separate number or axis length?).
- Transforming points shouldn’t be clumbsy.
- Need a better/cleaner conversion to matrix.
- How can we “multiply” (combine) two Axis Angle rotations??? …

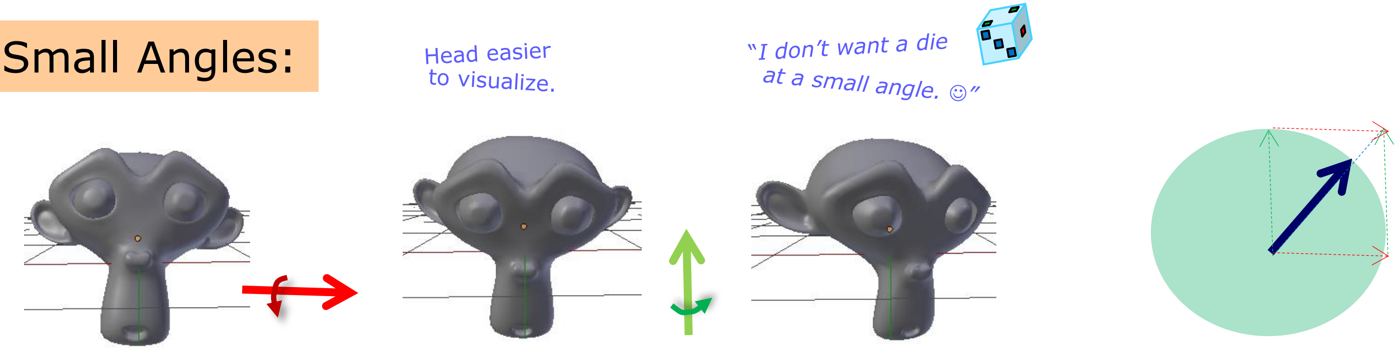
Combining Angle Axis Rotations

It's Tricky …

Small Angles:

Head easier to visualize.

"I don't want a die at a small angle. ☺"

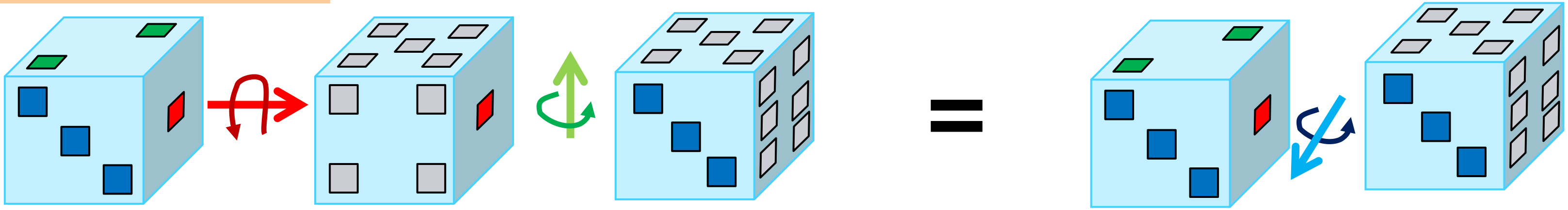


[1 0 0],10° then [0 1 0],10° ~= [1 1 0],14 °

Tilt, Turn, No Roll.

*"Result axis/angle is almost like vector addition on the **xy** plane"*

Larger Angles:

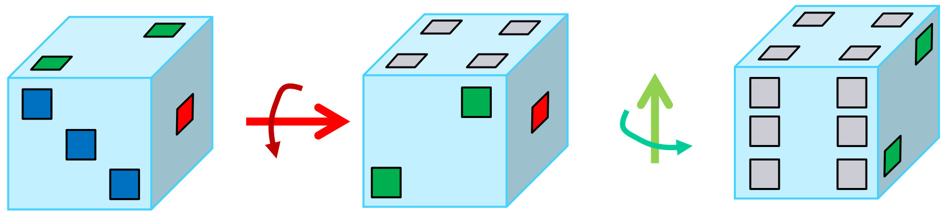


[100],180° then [010],180° = [0 0 1],180°

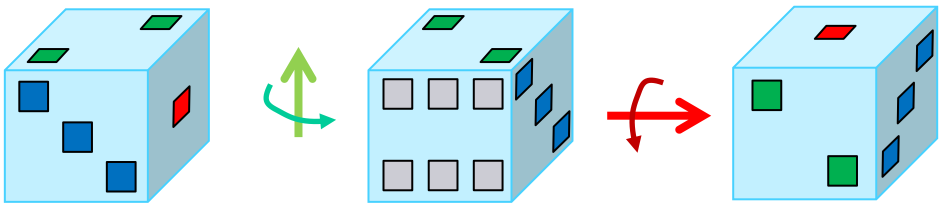
*"Hmmm, Combining **X** and **Y** somehow make **Z**"*

Combining Angle Axis Rotations

It's Tricky
Because…



[100],90 and [010],90 = …

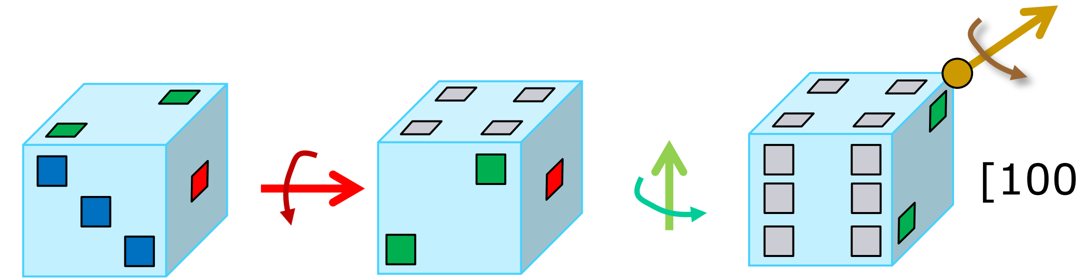


[010],90 and [100],90 = …

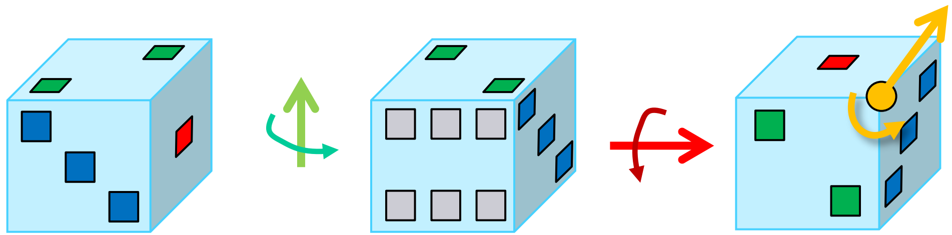
Order of rotations makes a difference…

Combining Angle Axis Rotations

It's Tricky Because ***Rotations are Tricky!***



[100],90 and [010],90 = [1 1 -1],120

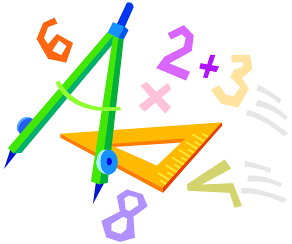


[010],90 and [100],90 = [1 1 1],120

Yikes. Is there any mathematics wizardry that can deal with this?

Quaternions – Mathematics of Rotations

Quaternions – Mathematics of Rotations



- Practical and Efficient (get the job done). Provides the machinery your program uses for rotational operations.
- Industry-wide standard algebraic system for dealing with rotations in 3D. (existing code, popular engines). *You'll need this*.
- Geometric Algebra encompass (and surpass) quaternions.
    - Still worth studying quats (stepping stone)
- A bit abstract (4D and complex numbers). Best to think visually/spatially.

Quaternions – not too complex ☺

- Like complex numbers $a+b\boldsymbol{i}$, but with 3 ⊥ sqrts of -1: $\boldsymbol{i},\boldsymbol{j},\boldsymbol{k}$
- $\boldsymbol{ii}=\boldsymbol{jj}=\boldsymbol{kk}=\boldsymbol{ijk}=-1$, so $\boldsymbol{ij}=\boldsymbol{k}$, $\boldsymbol{ji}=-\boldsymbol{k}$, $\boldsymbol{jk}=\boldsymbol{i}$, $\boldsymbol{ki}=\boldsymbol{j}$
- Numbers of the form: q= $a+b\boldsymbol{i}+c\boldsymbol{j}+d\boldsymbol{k}$ (math text notation)
- Isomorphic to Clifford Algebra R_{3+}: $q= a+b\boldsymbol{e}_{23}+c\boldsymbol{e}_{31}+d\boldsymbol{e}_{12}$
- In Practice: q= $x\boldsymbol{i}+y\boldsymbol{j}+z\boldsymbol{k}+w$ (graphics/gamedev convention)
- Quaternion multiplication:

$$\boldsymbol{ab} = \begin{array}{c} \left(+a_x b_w + a_y b_z - a_z b_y + a_w\, b_i\right)\boldsymbol{i} \\ +\left(-a_x b_z + a_y b_w + a_z b_x + a_w\, b_j\right)\boldsymbol{j} \\ +\left(+a_x b_y - a_y b_x + a_z b_w + a_w\, b_{\boldsymbol{k}}\right)\boldsymbol{k} \\ +\left(-a_x b_x - a_y b_y - a_z b_z + a_w\, b_w\right) \end{array}$$

Connection to Rotations may not be obvious yet…

Quaternions as Bivector,Scalar [**v**,w]

Equivalent to write quaternion as a bivector,scalar pair:

- Group the xyz elements into a 3D bivector **v** alongside w.

  Instead of: $[q_x, q_y, q_z, q_w]$, its now: $[\boldsymbol{q_v}, q_w]$

- Quaternion multiplication equivalent to:

$$\boldsymbol{ab} = [\boldsymbol{a_v}, a_w] * [\boldsymbol{b_v}, b_w] = [\underbrace{\boldsymbol{a_v} \times \boldsymbol{b_v}}_{\text{Cross Product}} + \boldsymbol{a_v} b_w + a_w \boldsymbol{b_v}, -\underbrace{\boldsymbol{a_v} \cdot \boldsymbol{b_v}}_{\text{Dot Product}} + a_w b_w]$$

some familiar operations

Unit Quaternions and Rotations

Use Quaternions on unit 4D hypersphere $(x^2+y^2+z^2+w^2 == 1)$:

- rotation/orientation with axis **a** and angle θ:

$$\boldsymbol{q} = \left[\boldsymbol{a}\sin\left(\frac{\theta}{2}\right), \cos\left(\frac{\theta}{2}\right)\right]$$

  - Length of bivector part proportional to sin of half of angle.
  - Value of scalar part *w* keeps quaternion at unit length (or cos of same half angle).

May be easier to visualize just using the (3D) bivector ***v*** component.

But its not a regular (Euclidean) 3-space.

Unit Quaternions and Rotations

Double Coverage:

Rotation around axis **a** and angle θ would produce the same result as rotation around axis -**a** and angle –θ.

Therefore, q and $-q$ represent the same rotation.

Inverse:

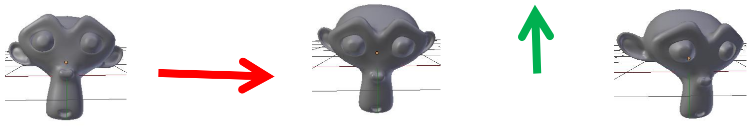
Rotation around axis -**a** and angle θ (or around **a** by - θ) would give the opposite rotation. Since q is of unit length just use conjugate:

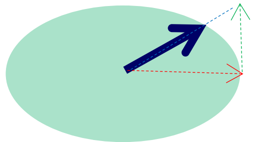
$$\boldsymbol{q}^{-1} = \boldsymbol{conj}(\boldsymbol{q}) = [-x, -y, -z, w] = [-v, w] = \left[-\boldsymbol{a}\sin\left(\frac{\theta}{2}\right), \cos\left(\frac{\theta}{2}\right)\right]$$

Examples Revisited with Quaternions:

Small Angles:

approx 10° on X
then 10° on Y





approx 15° on [1 1 -.1]

[0 0.1 0 0.99] * [0.1 0 0 0.99] ~= [0.1 0.1 -0.01 0.99]

Cross and dot Product near zero:

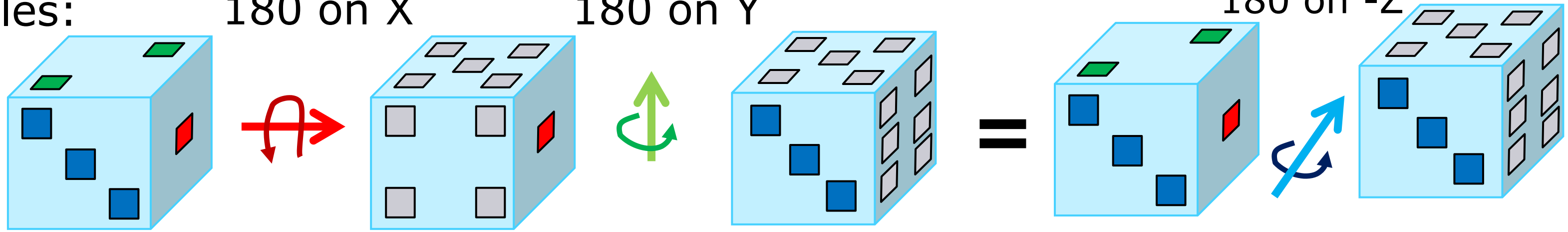
$$\boldsymbol{a}\,\boldsymbol{b} = [\boldsymbol{a_v} \times \boldsymbol{b_v} + \boldsymbol{a_v} b_w + a_w \boldsymbol{b_v} \;, -\boldsymbol{a_v} \cdot \boldsymbol{b_v} + a_w b_w]$$

Larger Angles:

180 on X

180 on Y

180 on -Z



[1 0 0 0]

[0 1 0 0]

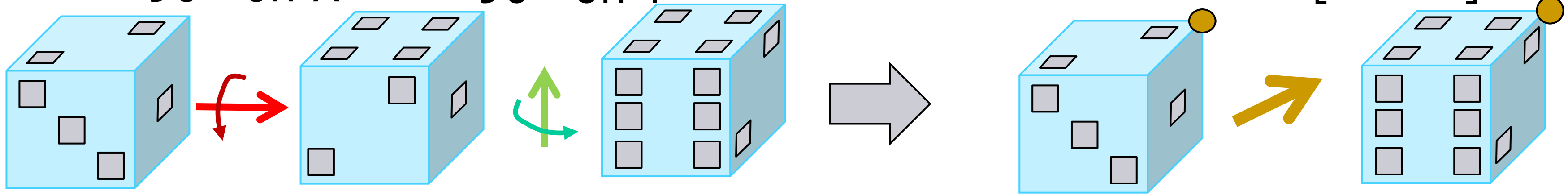
[0 1 0 0]*[1 0 0 0]=[0 0 -1 0]

$$[\boldsymbol{a_v} \times \boldsymbol{b_v} + \boldsymbol{a_v} b_w + a_w \boldsymbol{b_v} \;, -\boldsymbol{a_v} \cdot \boldsymbol{b_v} + a_w b_w] = \left[\underbrace{[0\;1\;0\,] \times [1\;0\;0]}_{} + 0 + 0 \;, 0 + 0\right] = \;[0\;0\;-1\;0]$$

Only Cross Product Matters here

Examples Revisited now with Quaternions:

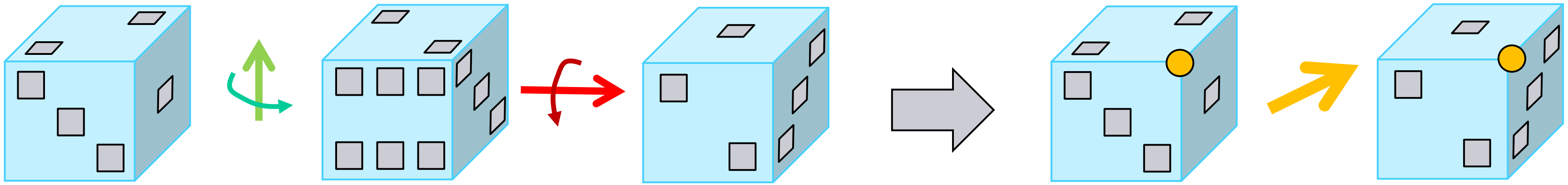
90° on X 90° on Y 120° on [1 1 -1]



$[.7\ \ 0\ \ 0\ \ .7]$ $[0\ \ .7\ \ 0\ \ .7]$

$$[0\ \ .7\ \ 0\ \ .7] * [.7\ \ 0\ \ 0\ \ .7] = [0.5\ \ 0.5\ \ -0.5\ \ 0.5]$$

90° on Y 90° on X 120° on [1 1 1]



$[0\ \ .7\ \ 0\ \ .7]$ $[.7\ \ 0\ \ 0\ \ .7]$

$$[0.7\ \ 0\ \ 0\ \ .7] * [0\ \ .7\ \ 0\ \ .7] = [.5\ .5\ .5\ .5]$$

Numerical values added just to see that the quaternion math indeed matches expectations.

Rotating Points/Vectors with Quaternions

Representation	Combine Rotations a,b	Rotate points or vectors (v)
Matrix:	$\mathbf{M}_b\ \mathbf{M}_a$	$\mathbf{M}\ \mathbf{v}$
Quaternion:	$\mathbf{q}_b\ \mathbf{q}_a$	$\mathbf{q}\ \mathbf{v}\ \mathbf{q}^{-1}$

- Matrix multiplication applies to both rotating points/vectors and other matrices.
- Rotate a point or vector ***v*** by treating it as a quaternion [***v***,0] and multiply by rotation and conjugate on the left and right sides respectively. Or use quaternion-to-matrix conversion.

qvq^{-1}

$ab = [\boldsymbol{a_v} \times \boldsymbol{b_v} + \boldsymbol{a_v} b_w + a_w \boldsymbol{b_v}\ , -\boldsymbol{a_v} \cdot \boldsymbol{b_v} + a_w b_w]$

quaternion multiplication (bivector-scalar style)

just multiply these three 4D numbers

$$\boldsymbol{qvq}^{-1} = \left[\mathbf{a}\sin\frac{\theta}{2}, \cos\frac{\theta}{2}\right][\boldsymbol{v}, 0]\left[-\mathbf{a}\sin\frac{\theta}{2}, \cos\frac{\theta}{2}\right]$$

$$= \left[\sin\frac{\theta}{2}\mathbf{a} \times \boldsymbol{v} + \cos\frac{\theta}{2}\ \boldsymbol{v}, -\ \sin\frac{\theta}{2}\boldsymbol{a}\cdot\boldsymbol{v}\right]\left[-\mathbf{a}\sin\frac{\theta}{2}\ , \cos\frac{\theta}{2}\right]$$

Not too bad so far

$\mathbf{a} \times \boldsymbol{v} \cdot \mathbf{a} == 0$

Yikes!

$$= \left[\sin\frac{\theta}{2}\mathbf{a} \times \boldsymbol{v} \times -\mathbf{a}\ \sin\frac{\theta}{2} + \cos\frac{\theta}{2}\ \boldsymbol{v} \times -\mathbf{a}\ \sin\frac{\theta}{2} + \cos\frac{\theta}{2}\ \sin\frac{\theta}{2}\mathbf{a} \times \boldsymbol{v} + \cos\frac{\theta}{2}\cos\frac{\theta}{2}\ \boldsymbol{v} + -\mathbf{a}\sin\frac{\theta}{2}\ \sin\frac{\theta}{2}(-\boldsymbol{a}\cdot\boldsymbol{v}), -\boldsymbol{a}\cdot\boldsymbol{v}\ \sin\frac{\theta}{2}\cos\frac{\theta}{2} + \ \sin\frac{\theta}{2}\mathbf{a} \times \boldsymbol{v}\cdot\mathbf{a}\ \sin\frac{\theta}{2} + \cos\frac{\theta}{2}\ \boldsymbol{v}\cdot\mathbf{a}\ \sin\frac{\theta}{2}\right]$$

cancel out

$$= \left[-\sin^2\frac{\theta}{2}\mathbf{a} \times \boldsymbol{v} \times \mathbf{a} + \underbrace{2\cos\frac{\theta}{2}\ \sin\frac{\theta}{2}}_{\sin\theta}\mathbf{a} \times \boldsymbol{v} + \cos^2\frac{\theta}{2}\ \boldsymbol{v} + \sin^2\frac{\theta}{2}(\boldsymbol{a}\cdot\boldsymbol{v})\mathbf{a},\qquad 0\right]$$

$$= \left[-\sin^2\frac{\theta}{2}\mathbf{a} \times \boldsymbol{v} \times \mathbf{a} + \ \sin\theta\,\mathbf{a} \times \boldsymbol{v} + \cos^2\frac{\theta}{2}\ \boldsymbol{v} + \sin^2\frac{\theta}{2}(\boldsymbol{a}\cdot\boldsymbol{v})\mathbf{a},\qquad 0\right]$$

use some half angle formulas

$$= \left[\cos^2\frac{\theta}{2}(\mathbf{a} \times \boldsymbol{v} \times \mathbf{a}) -\ \sin^2\frac{\theta}{2}\mathbf{a} \times \boldsymbol{v} \times \mathbf{a} + \ \sin\theta\,\mathbf{a} \times \boldsymbol{v} + \cos^2\frac{\theta}{2}(\boldsymbol{a}\cdot\boldsymbol{v})\mathbf{a} + \sin^2\frac{\theta}{2}(\boldsymbol{a}\cdot\boldsymbol{v})\mathbf{a},\qquad 0\right]$$

$\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2} = \cos\theta$

$\cos^2 + \sin^2 = 1$

$$= \ [\cos\theta\,(\mathbf{a} \times \boldsymbol{v} \times \mathbf{a}) + \sin\theta\,(\mathbf{a} \times \boldsymbol{v}) + (\boldsymbol{a}\cdot\boldsymbol{v})\mathbf{a}\ , 0]$$

after simplifying

qvq^{-1}: Rotating Points/Vectors

$$q = \left[\mathbf{a}\ \sin\frac{\theta}{2}, \cos\frac{\theta}{2}\right]$$

Three Orthogonal Vectors

*Portion along **a** stays the same*

$$qvq^{-1} = [\cos\theta\,(\mathbf{a}\times \boldsymbol{v}\times\mathbf{a}) + \sin\theta\,(\mathbf{a}\times\boldsymbol{v}) + (\boldsymbol{a}\cdot\boldsymbol{v})\mathbf{a}\ , 0]$$

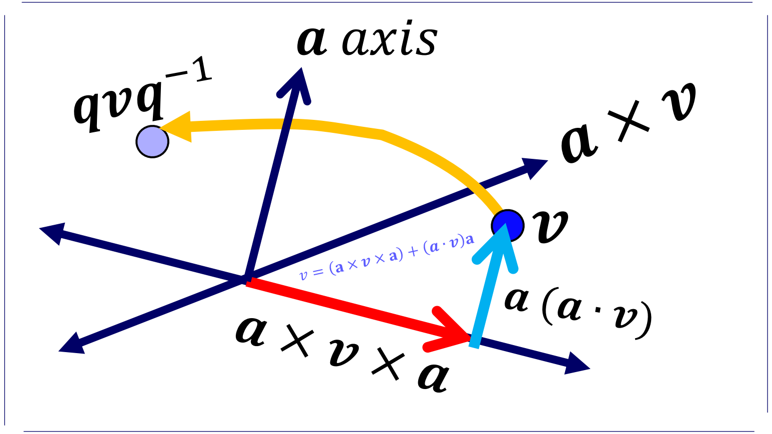
sum weighted by sin and cos

Quaternion multiplication

qvq^{-1} transforms ***v*** by rotation ***q***

v lies in plane of 2 of these basis vectors:

$$v = (\mathbf{a}\times v\times\mathbf{a}) + (\boldsymbol{a}\cdot v)\mathbf{a}$$



Applications

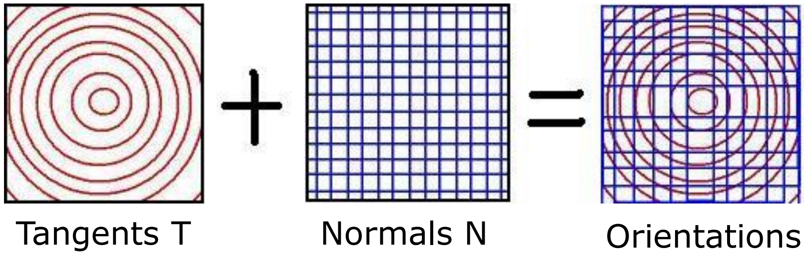
Quaternions can replace most Rotation Matrices

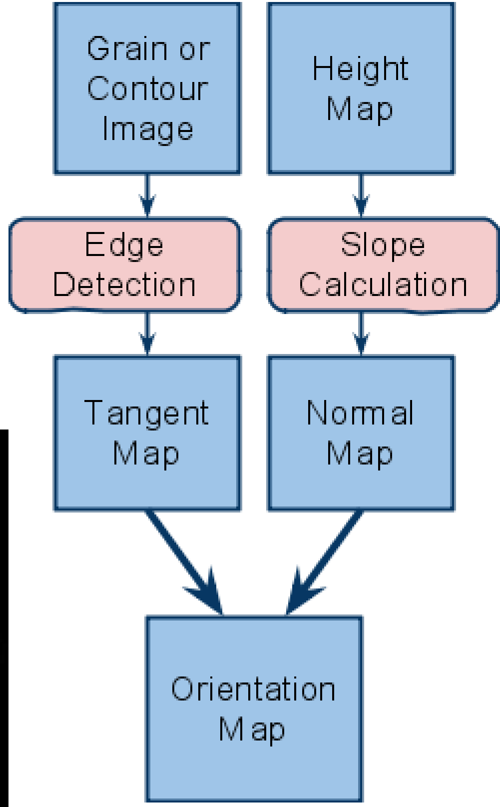
- Cameras or any general objects with position and orientation.
- Rigid Bodies - physics engines mostly use vec/quat pairs
- Vertex buffers instead of tangent,bitangent,normal can use:

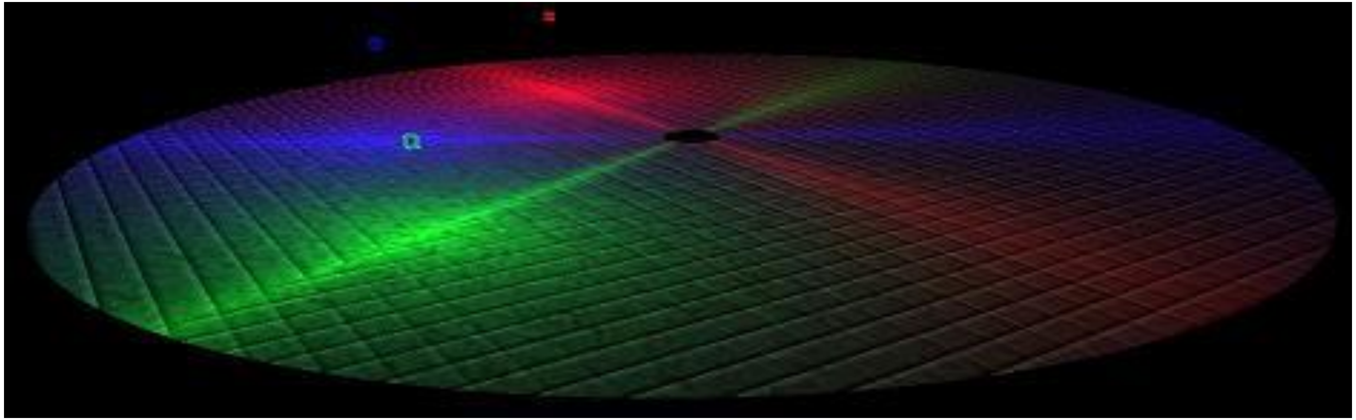
```
struct Vertex {
    float3 position;    // location in mesh reference frame
    float4 orientation; // quaternion tangent space basis
    float2 texcoord;    // uv's
    ...
```

Orientation Map

- Extention of normalmap
- rgba encodes orientation.







Disc with specular $(T \cdot L)$ and diffuse $(N \cdot L)$

Disclaimer: just curiosity research, not sure how useful.

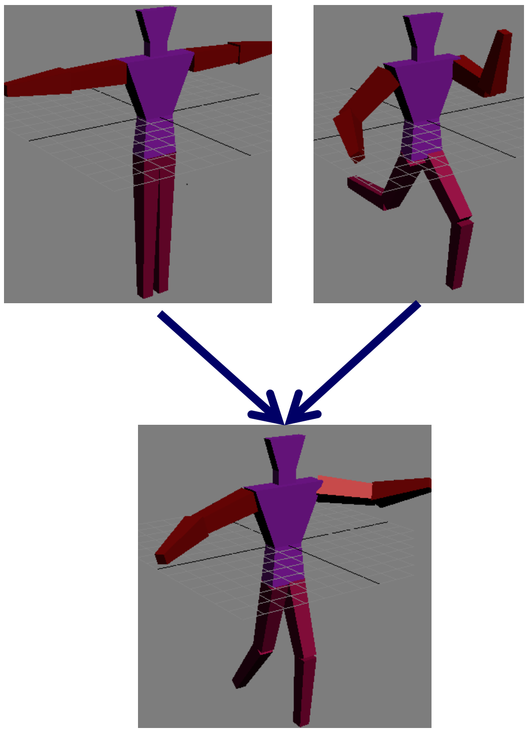
SLERP – Spherical Linear Interpolation

- Smooth transition between orientations q_0, q_1
  - Double Coverage Issue: Use $-q_1$ instead of q_1 if closer to q_0
- Normalized Lerp (nlerp) often sufficient

$$q_t = normalize(q_0(1-t) + q_1(t))$$

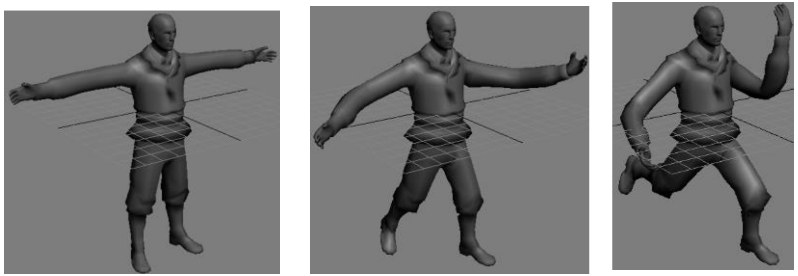
- Used by animation systems (blend keyframes)

Key 0 Key 1



NLERP 0.5

Resulting Skinned Animation



t=0 t=0.5 t=1

Quats – *they do Addition too…*

Proof it's the same:

$$\lim_{(\|\omega\|\,dt)\to 0} s \to \left[\frac{\omega}{2}dt, 1\right]$$

$$s * q_t = [0001] * q_t + \left[\frac{\omega}{2}dt, 0\right] * q_t$$

$$s * q_t = q_t + \left[\frac{\omega}{2}, 0\right] * q_t\, dt$$

Updating state to the next time step.

- Position: $p_{t+dt} = p_t + velocity * dt$
- Orientation (spin ω):

Could Build a Quat for Multiplication

$$s = \left[\frac{\omega}{\|\omega\|}\sin(\frac{\|\omega\|\,dt}{2}), \cos(\frac{\|\omega\|\,dt}{2})\right]$$

$$q_{t+dt} = s * q_t$$

Or Add Derivative

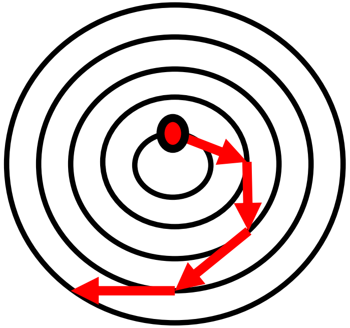
$$q_{t+dt} = q_t + \frac{dq}{dt}\,dt$$

$$q_{t+dt} = q_t + \frac{\omega}{2} q_t\, dt$$

Ok but why?…

Quat Application: Time Integration (no drift)

Spin ω_t is not constant!!



$$q_{t+dt} = q_t + \frac{\omega(q_t)}{2} * q_t * dt$$

Forward Euler

only looks at starting spin

more quat additions

$$k_1 = \frac{\omega(q_t)}{2} * q_t$$

$$k_2 = \frac{\omega(q_t + k_1 * dt/2)}{2} * (q_t + k_1 * \frac{dt}{2})$$

$$k_3 = \frac{\omega(q_t + k_2 * dt/2)}{2} * (q_t + k_2 * \frac{dt}{2})$$

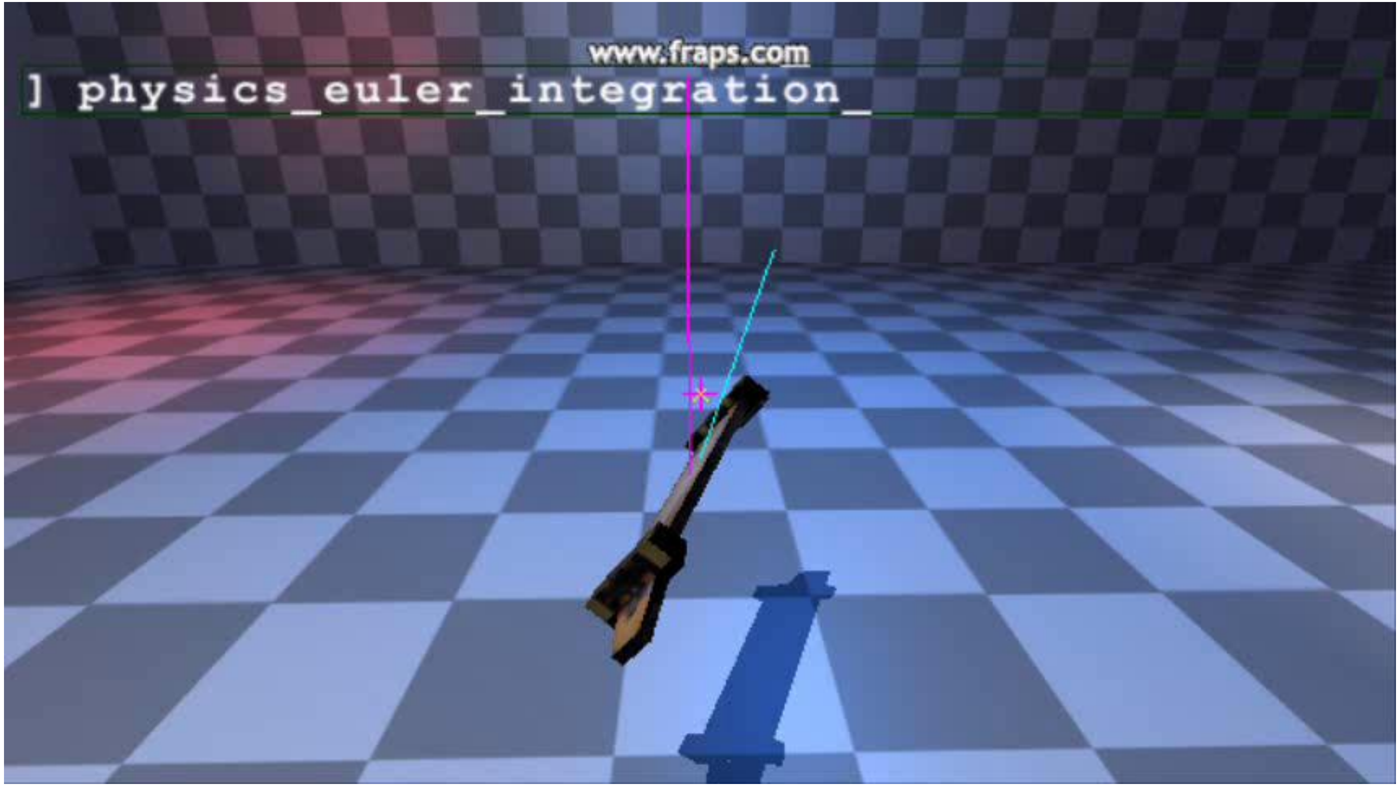
$$k_4 = \frac{\omega(q_t + k_3 * dt)}{2} * (q_t + k_3 * dt)$$

$$q_{t+dt} = q_t + k_1 * \frac{dt}{6} + k_2 * \frac{dt}{3} + k_3 * \frac{dt}{3} + k_4 * \frac{dt}{6}$$

Runge Kutta

Takes samples over the timestep

Orientation Updates (Euler vs RK4)



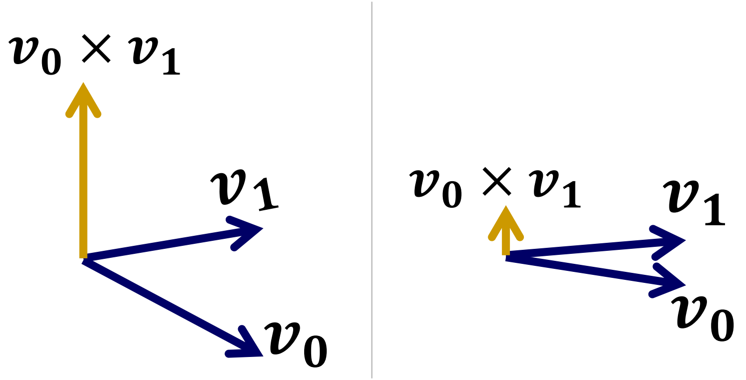
Forward Euler:
- Spin drifts toward principle axis
- Energy gained

Runge Kutta
- Spin orbits as expected
- Energy stays constant

Watch GDC 2013 Math Tutorial for full explanation of inertia tensor, time integration, angular momentum, rk4, …

Rotation that takes one direction $\boldsymbol{v}_0$ to another $\boldsymbol{v}_1$

Cross product to find axis $\boldsymbol{a}$



When $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$ get close,
$\boldsymbol{a} = \boldsymbol{v}_0 \times \boldsymbol{v}_1$ becomes small
$\boldsymbol{d} = \boldsymbol{v}_0 \cdot \boldsymbol{v}_1$ goes to 1.

Ignore $\boldsymbol{v}_0 = -\boldsymbol{v}_1$ case for now

What if $\|\boldsymbol{a}\|$ was 0

Less stable when d ~ 1

$$\boldsymbol{q} = \left[\frac{\boldsymbol{a}}{\|\boldsymbol{a}\|}\sin\left(\frac{\operatorname{acos}(d)}{2}\right), \cos\left(\frac{\operatorname{acos}(d)}{2}\right)\right]$$

using half angle formulas

$$\boldsymbol{q} = \left[\frac{\boldsymbol{a}}{\sqrt{2(1+d)}}, \left(\frac{\sqrt{2(1+d)}}{2}\right)\right]$$

GA style – geometric product produces rotation versor

$$\text{Let: } \boldsymbol{v}_{mid} = \frac{\boldsymbol{v}_0+\boldsymbol{v}_1}{\|\boldsymbol{v}_0+\boldsymbol{v}_1\|}$$

$$\boldsymbol{q} = [\boldsymbol{v}_0 \times \boldsymbol{v}_{mid}, \boldsymbol{v}_0 \cdot \boldsymbol{v}_{mid}]$$

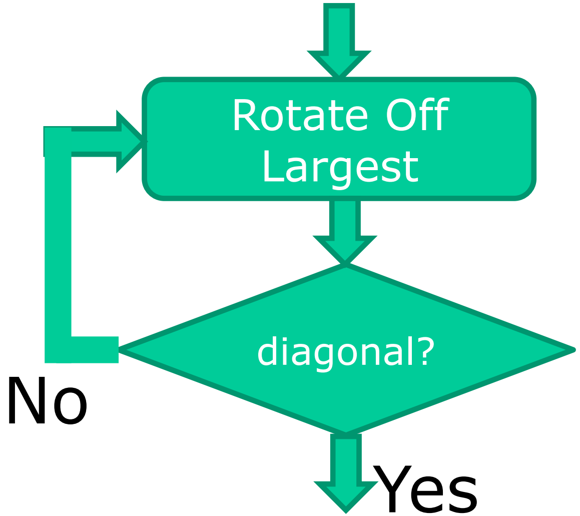
Diagonalization of Symmetric Matrices

$$\begin{bmatrix} a & f & e \\ f & b & d \\ e & d & c \end{bmatrix}$$

For symmetric matrix ***S*** find ***D,R***:

$$\boldsymbol{D} = \boldsymbol{R}\,\boldsymbol{S}\,\boldsymbol{R}^{-1}$$

- Iterative approach [Jacobi 1800s].
- Algorithm can accumulate directly into Matrix or a Quaternion (3D).



Eigenvalues are entries of diagonal part.

If not all equal, this may be interpreted as an orientation for the matrix in some contexts.

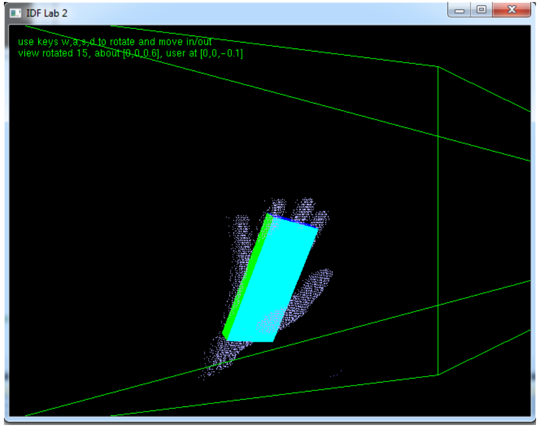
$$[x\ y\ z\ w], \begin{bmatrix} a' & 0 & 0 \\ 0 & b' & 0 \\ 0 & 0 & c' \end{bmatrix}$$

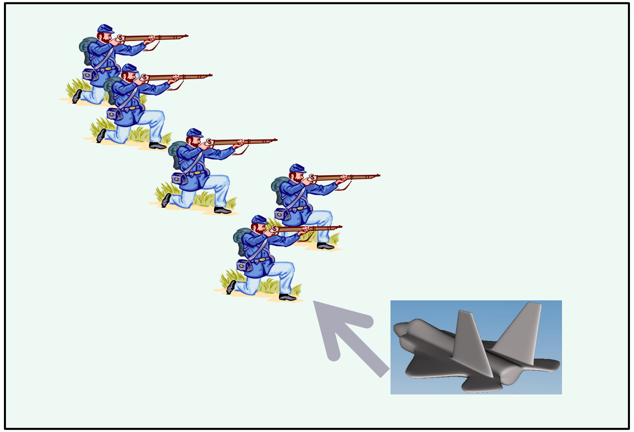
"Orientations may show up in new interesting places"

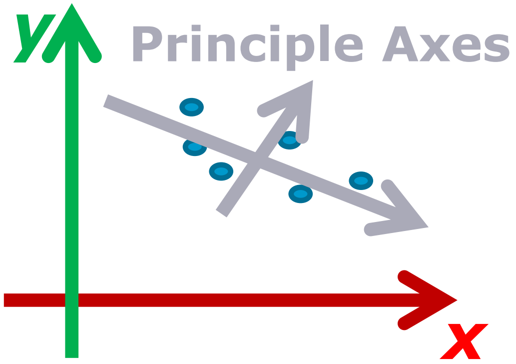
Orientation of a Point Cloud

- Compute covariance
- Diagonalize to get orientation.
- Permute by eigenvalues for long,med,short axes.

$$cov = \sum vv^T = \sum \begin{bmatrix} {v_x}^2 & v_x v_y & v_x v_z \\ v_y v_x & {v_y}^2 & v_y v_z \\ v_z v_x & v_z v_y & {v_z}^2 \end{bmatrix}$$







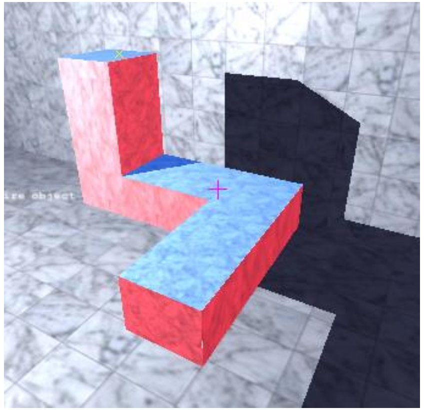
UI: Data from Depth Sensor

AI: Optimal bombing run.

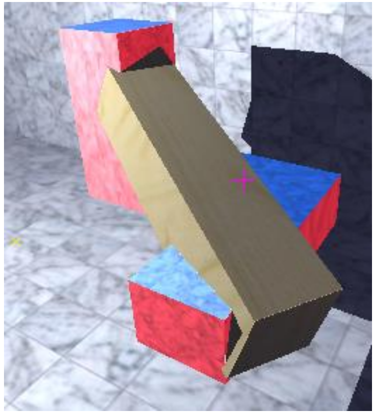
Visualize Inertia Properties

To debug physics behavior of rigid body:

- Diagonalize Inertia Tensor (symmetric matrix)
- Draw box over object with resulting orientation
- Eigenvalues are box dimensions



Irregular Shape



Inertia Overlay

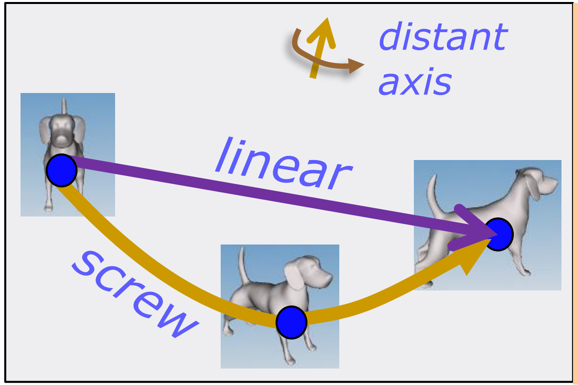
Dual Quaternions

- Add a $\sqrt{0}$ or ε, $\varepsilon^2 = 0$
- $q = [x\boldsymbol{i},\ y\boldsymbol{j},\ z\boldsymbol{k},\ w,\ x'\boldsymbol{i\varepsilon},\ y'\boldsymbol{j\varepsilon},\ z'\boldsymbol{k\varepsilon},\ w'\boldsymbol{\varepsilon}]$
- Put half translation $\boldsymbol{t}$ in dual part
$$\boldsymbol{t}'' = [0,0,0,1, \quad tx/2, ty/2, tz/2, 0]$$
- Extend rotation $\boldsymbol{r}$ to dual quat
$$\boldsymbol{r}'' = [\boldsymbol{r}, \quad 0,0,0,0]$$
- Multiply trans and rot dual quaternions
$$\boldsymbol{q}'' = \boldsymbol{t}''\boldsymbol{r}''$$

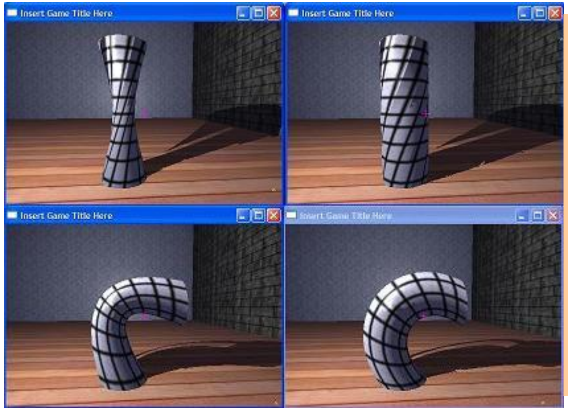
Rotation and Trans in a single 8D number $\boldsymbol{q}''$

To be continued (in Gino's IK session) …

Dual Quat Screw Motion



SLERP



SKINNING

Matrix Dual Quat

Working with Rotations - Conclusion

- Rotations can be tricky (*don't blame math*)
- Matrices work
- Quaternions work, more concise, more uses
- Be Aware, Be Precise:
  - who to multiply
  - what order to use
  - when to invert

Now go and do cool 3D stuff ☺

Q & A

Raise your hand if you have any questions now!