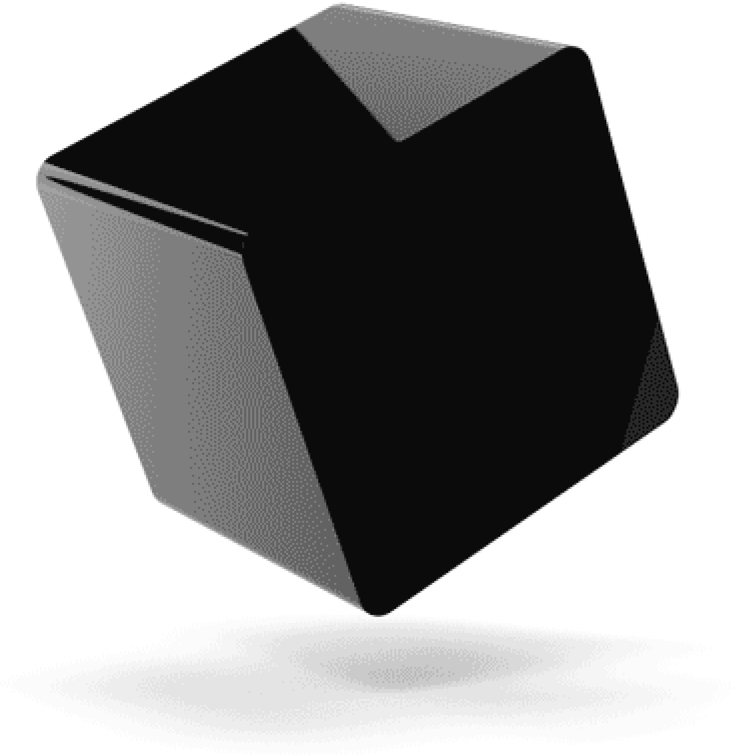
GRK 8

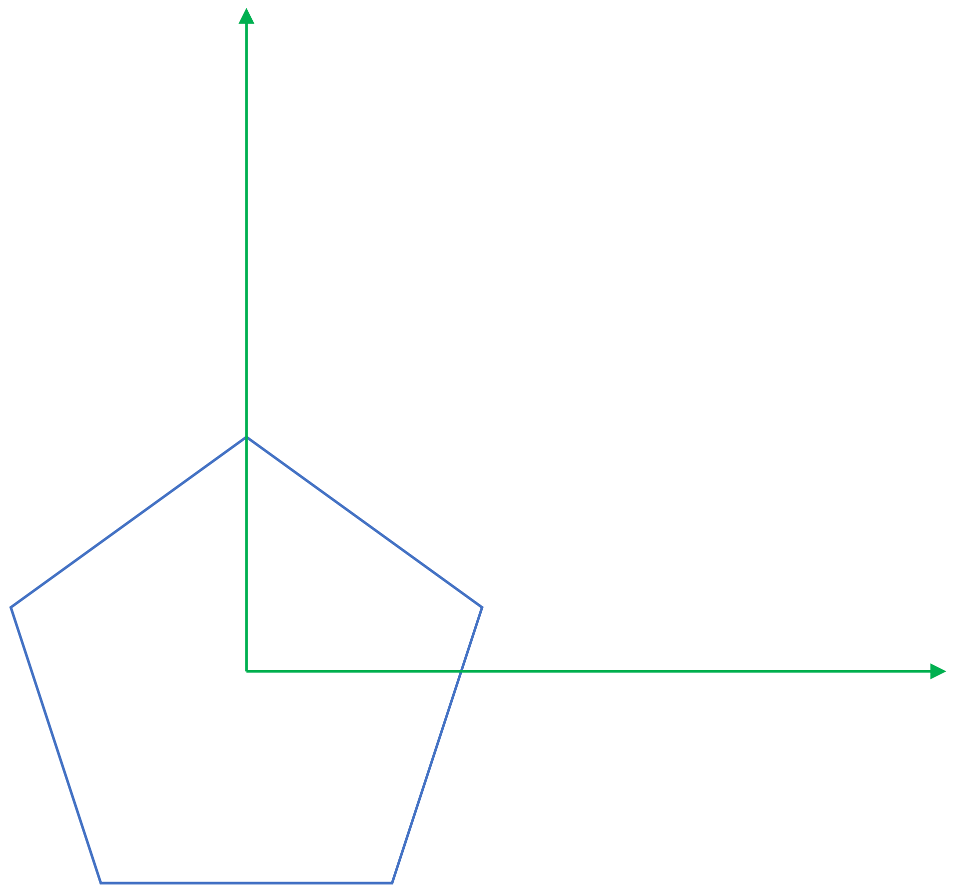
Dr Wojciech Palubicki

Rotation



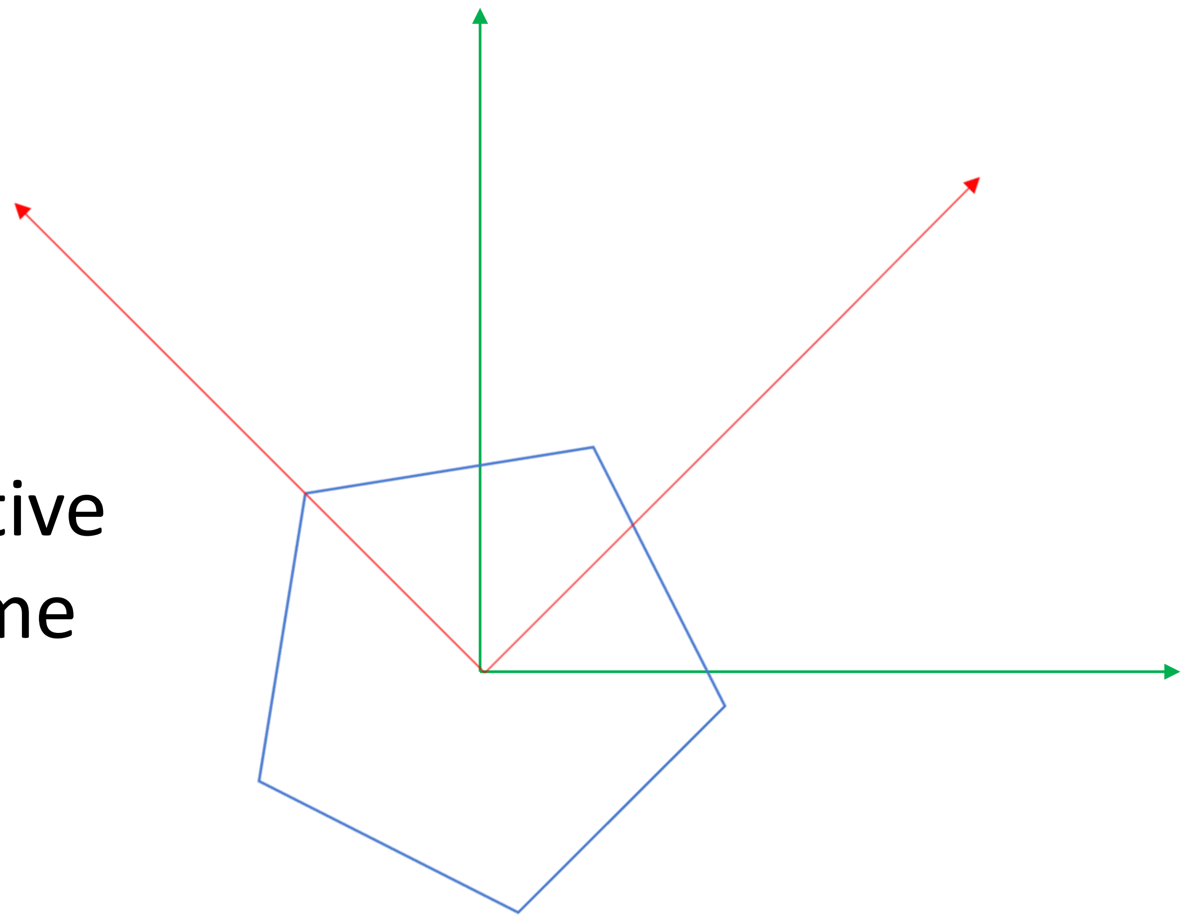
Defining Rotation

Reference frame



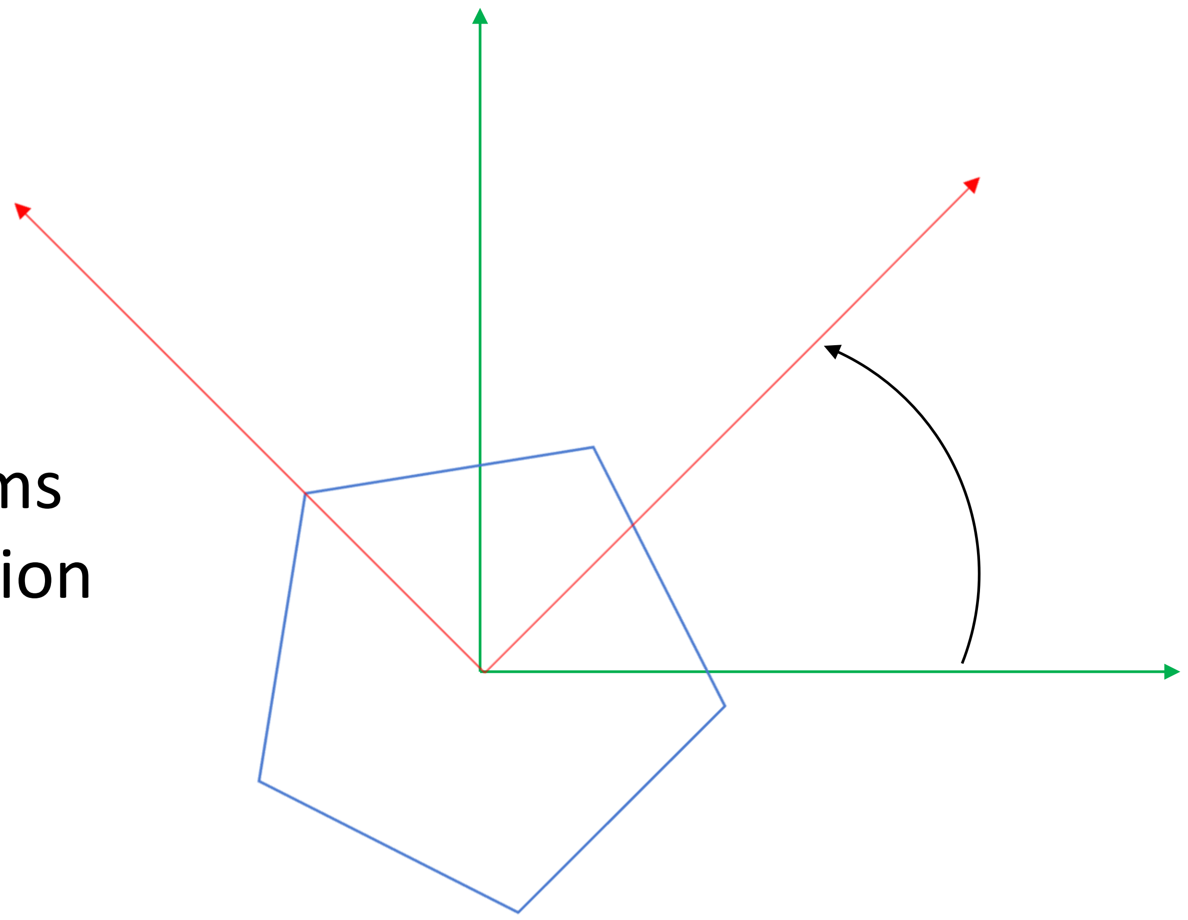
Defining Rotation

Orientation relative to reference frame

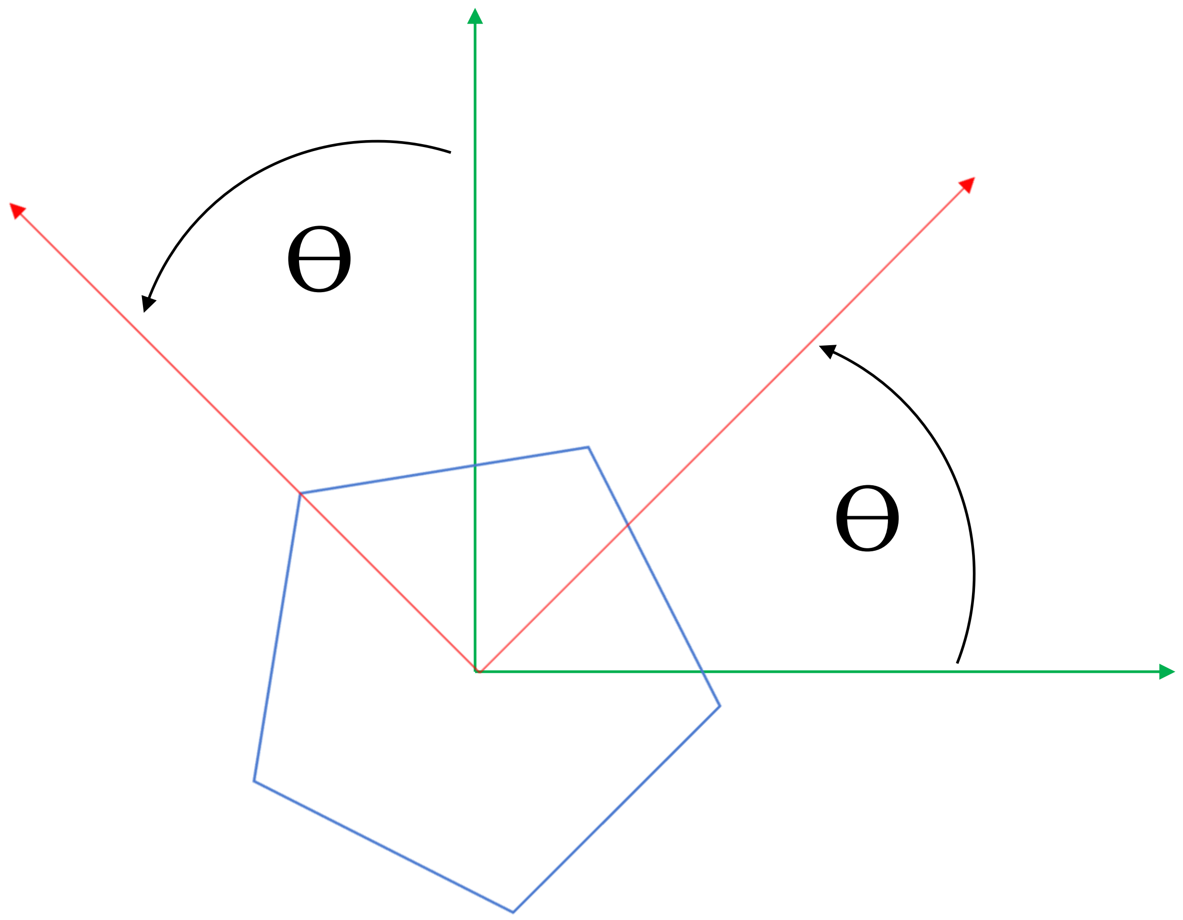


Defining Rotation

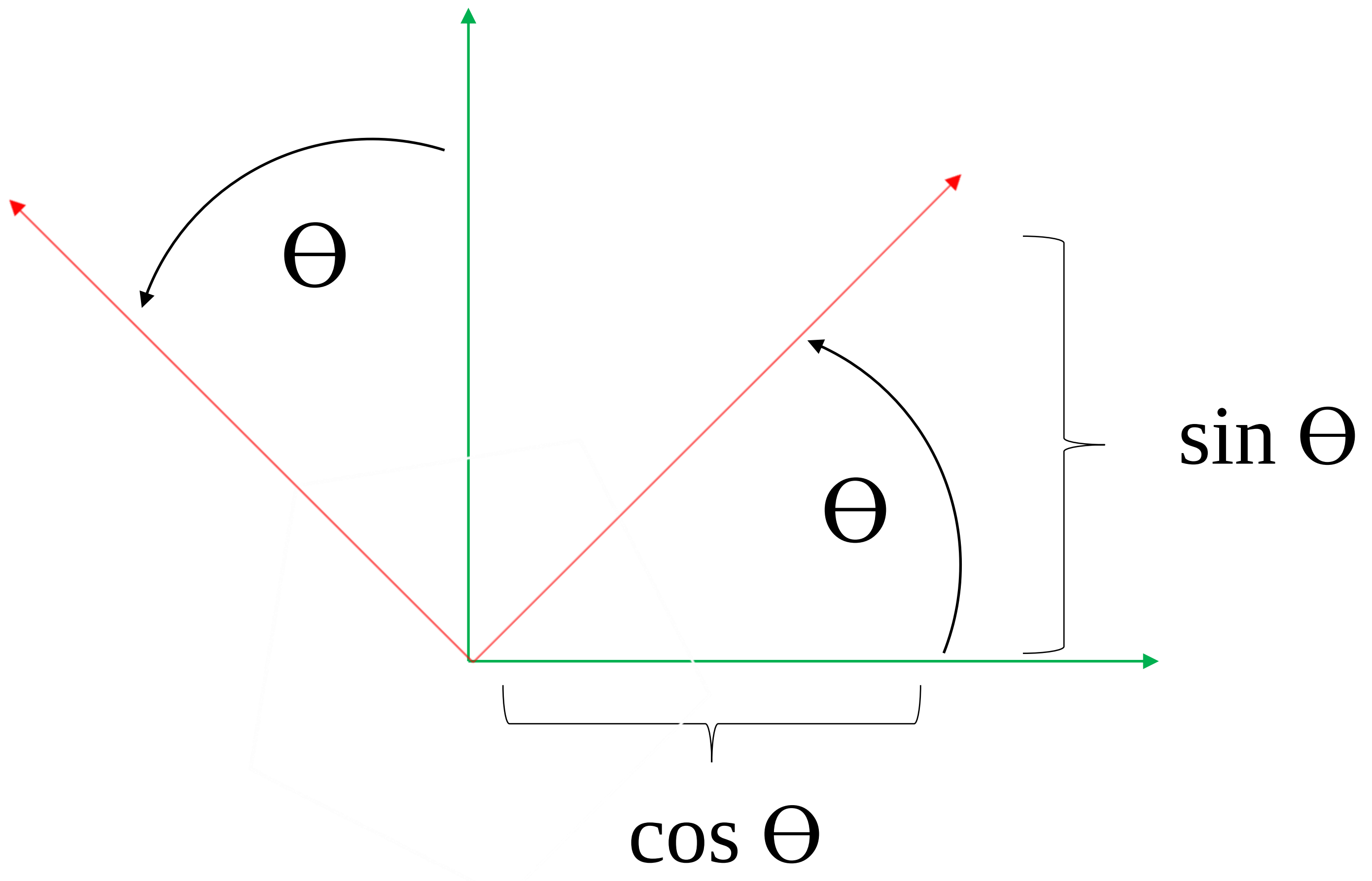
Rotation transforms from one orientation to another



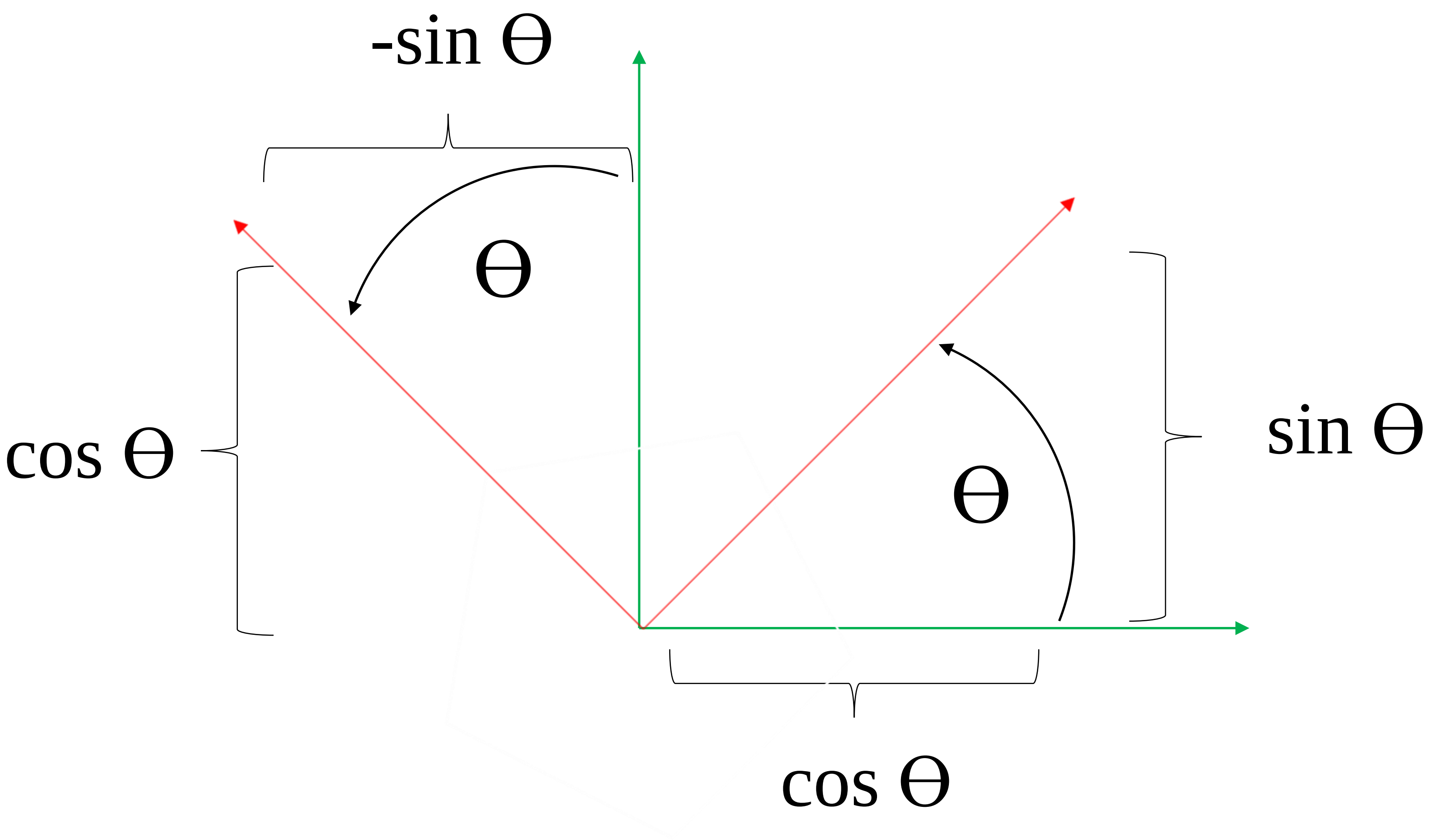
Rotation Angle (2D)



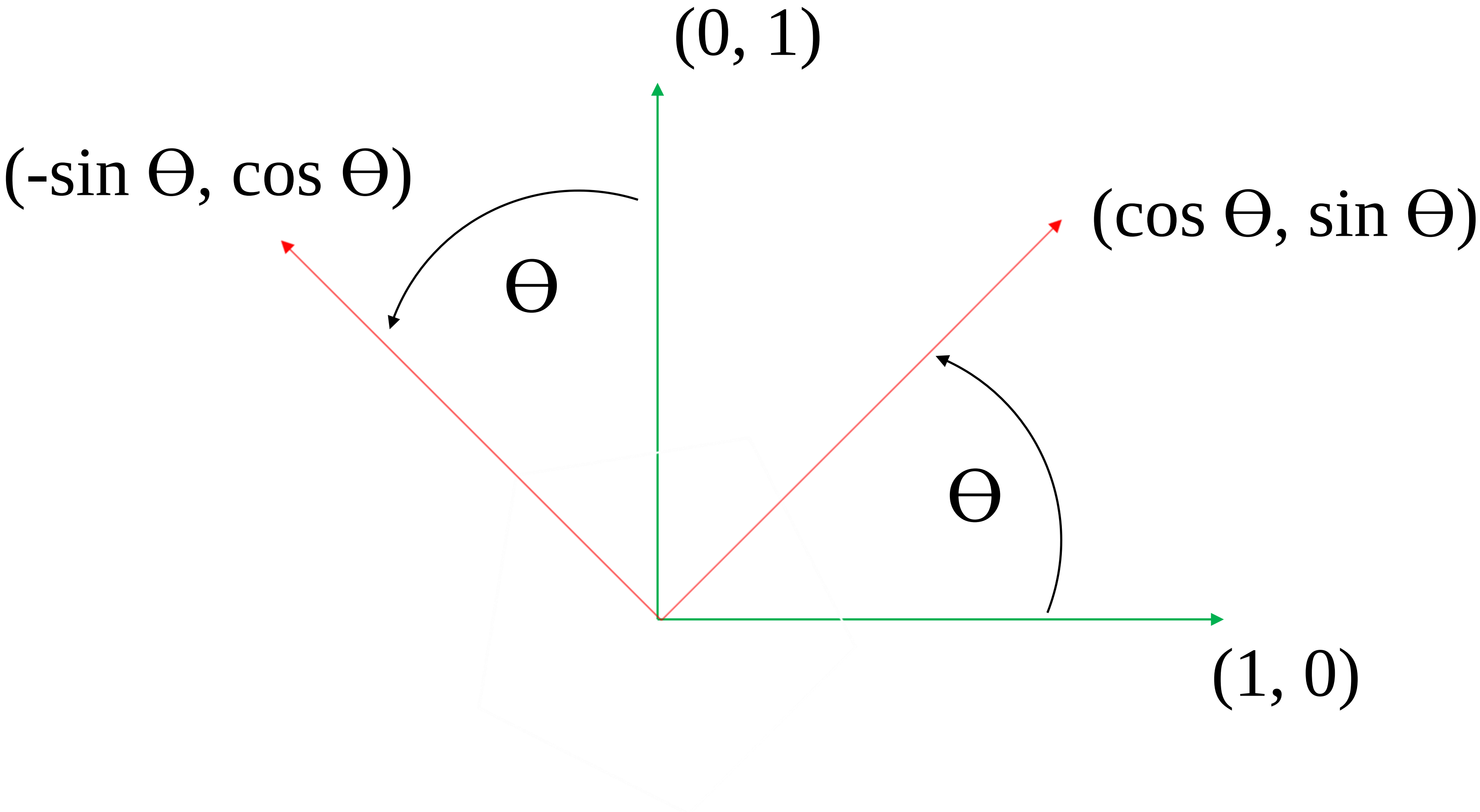
Rotation Angle (2D)



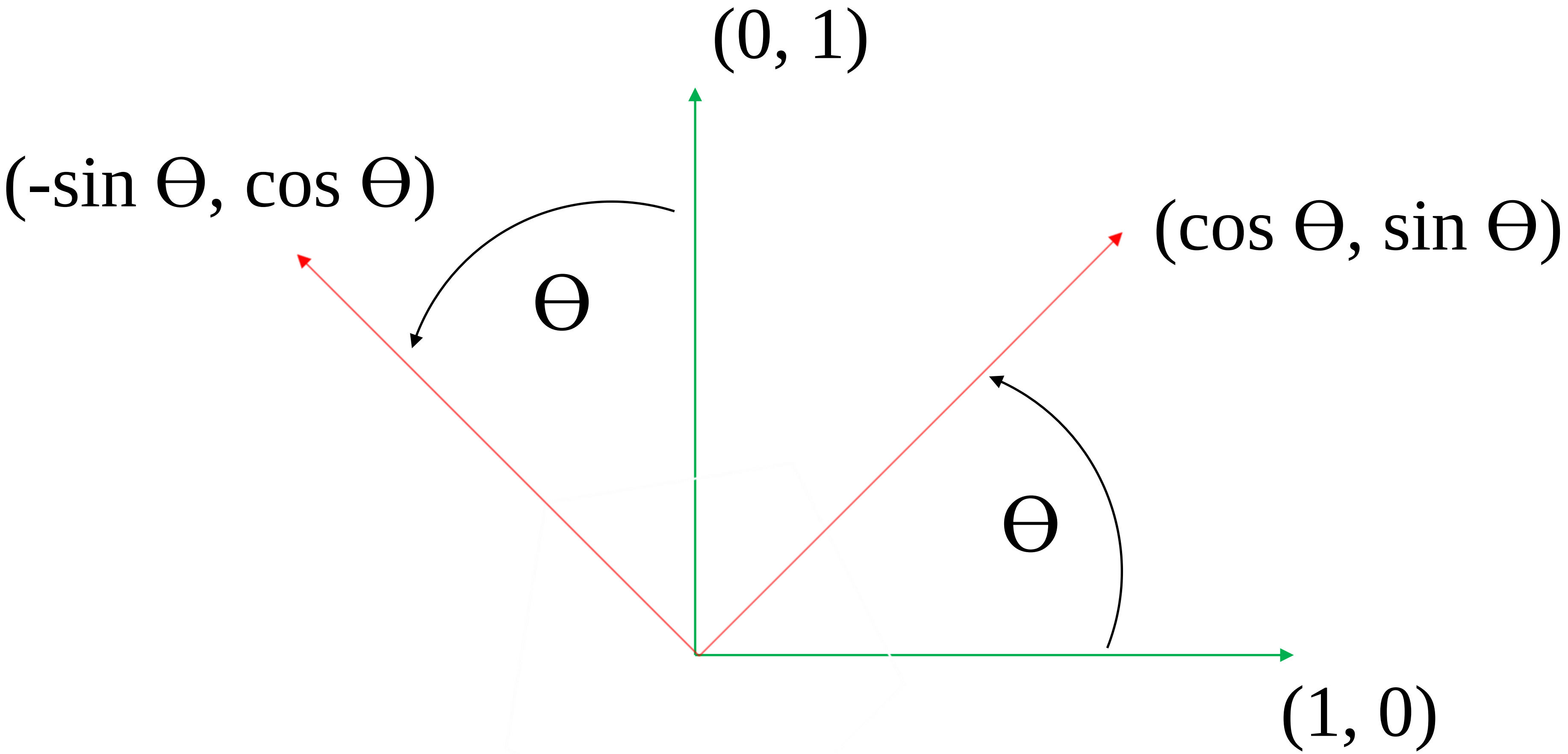
Rotation Angle (2D)



Rotation Angle (2D)



Transformation



$(x, y) \rightarrow (x \cos \Theta - y \sin \Theta, x \sin \Theta + y \cos \Theta)$

Orientation vs Rotation

- **Orientation** of an object is described relative to some reference frame
- A **rotation** changes an object from one **orientation** to another

2D Rotation Matrix

- Transformation
  - (1, 0) → (cos Θ, sin Θ)
  - (0, 1) → (-sin Θ, cos Θ)

- Rotation matrix
  - $\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$

3D Rotation Matrices

- Rotation Matrix attributes:
  - Orthogonal
    - Basis vectors are unit length
    - Dot products of basis vectors are zero
    - $M^{-1} = M^T$
    - $M^T M = \mathbf{I}$
  - Determinant is 1

3D Matrices $R_x R_y R_z$

$$R_X R_Y R_Z$$
$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos A & -\sin A \\ 0 & \sin A & \cos A \end{pmatrix} \begin{pmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{pmatrix} \begin{pmatrix} \cos C & -\sin C & 0 \\ \sin C & \cos C & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3D Matrices $R_x R_y R_z$

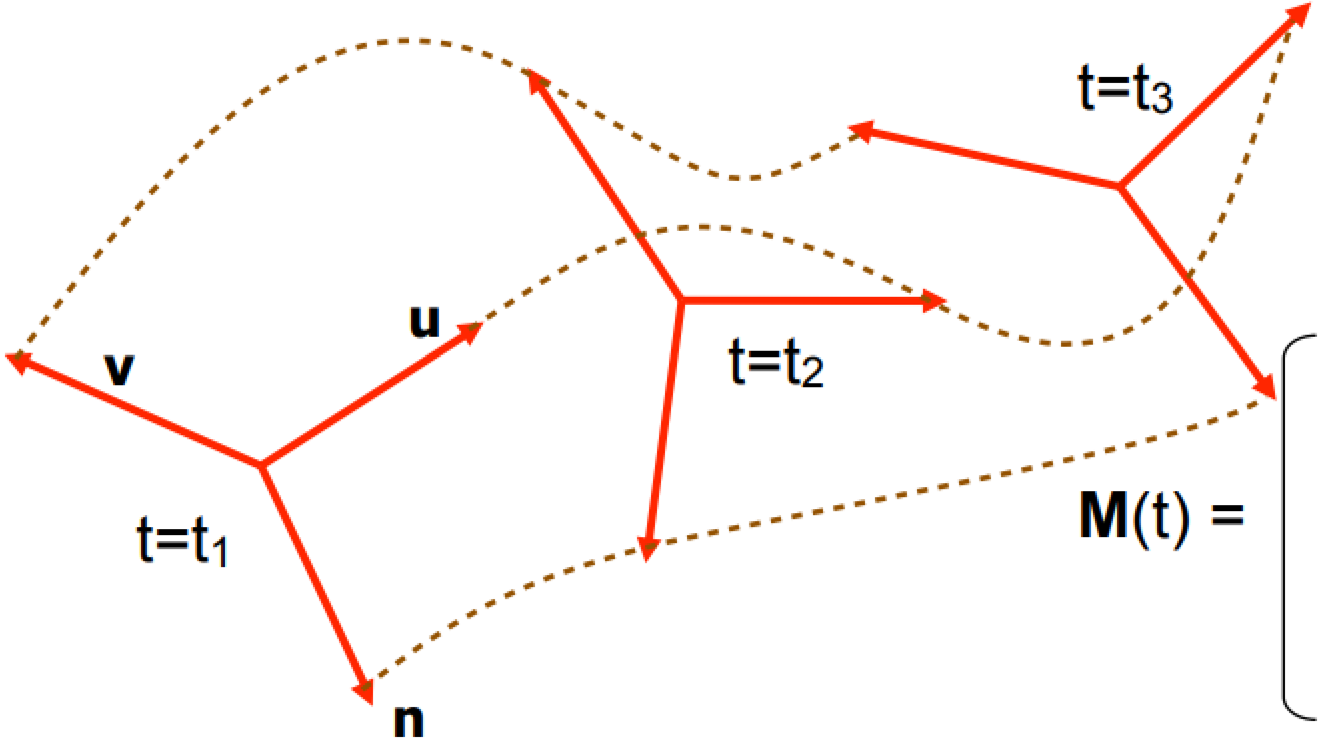
$$
\begin{aligned}
&R_X R_Y R_Z \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos A & -\sin A \\ 0 & \sin A & \cos A \end{pmatrix} \begin{pmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{pmatrix} \begin{pmatrix} \cos C & -\sin C & 0 \\ \sin C & \cos C & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos B & 0 & \sin B \\ \sin A \sin B & \cos A & -\sin A \cos B \\ -\cos A \sin B & \sin A & \cos A \cos B \end{pmatrix} \begin{pmatrix} \cos C & -\sin C & 0 \\ \sin C & \cos C & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos B \cos C & -\cos B \sin C & \sin B \\ \sin A \sin B \cos C + \cos A \sin C & -\sin A \sin B \sin C + \cos A \cos C & -\sin A \cos B \\ -\cos A \sin B \cos C + \sin A \sin C & \cos A \sin B \sin C + \sin A \cos C & \cos A \cos B \end{pmatrix}
\end{aligned}
$$

Pitch, Yaw, Roll

Ideal Orientation Format

- Represent 3 degrees of freedom with minimum number of values
- Allow concatenations of rotations
- Math should be simple and efficient
  - concatenation
  - rotation
  - interpolation

Interpolation?



$$\mathbf{M}(t) = \begin{bmatrix} u_x(t) & u_y(t) & u_z(t) \\ v_x(t) & v_y(t) & v_z(t) \\ n_x(t) & n_y(t) & n_z(t) \end{bmatrix}$$

Interpolating Matrices

- Say we interpolate halfway between each element

$$0.5\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}+0.5\begin{pmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}=\begin{pmatrix}0.5 & -0.5 & 0\\ 0.5 & 0.5 & 0\\ 0 & 0 & 1\end{pmatrix}$$

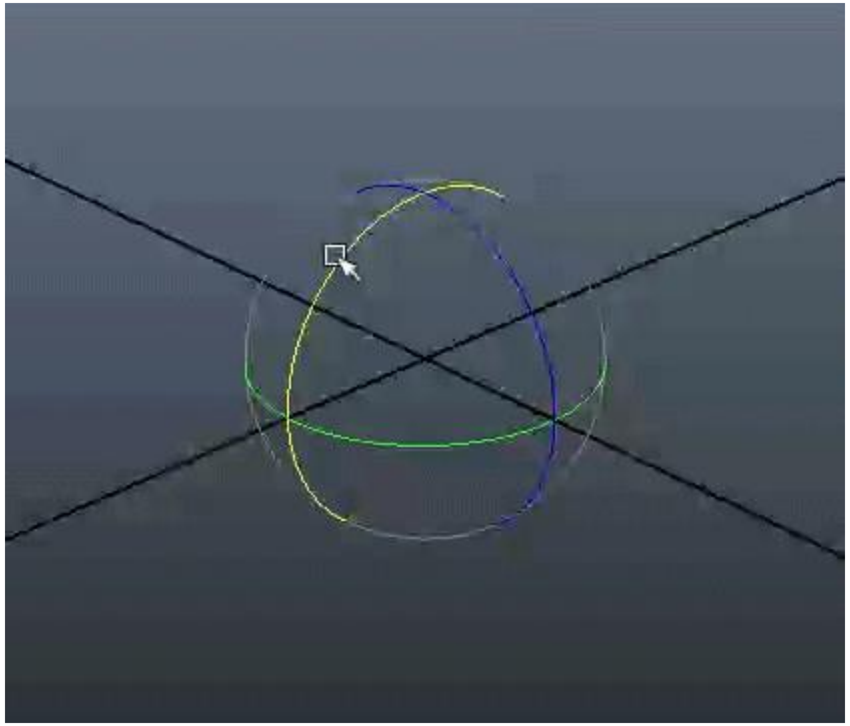
Interpolating Matrices

- Say we interpolate halfway between each element

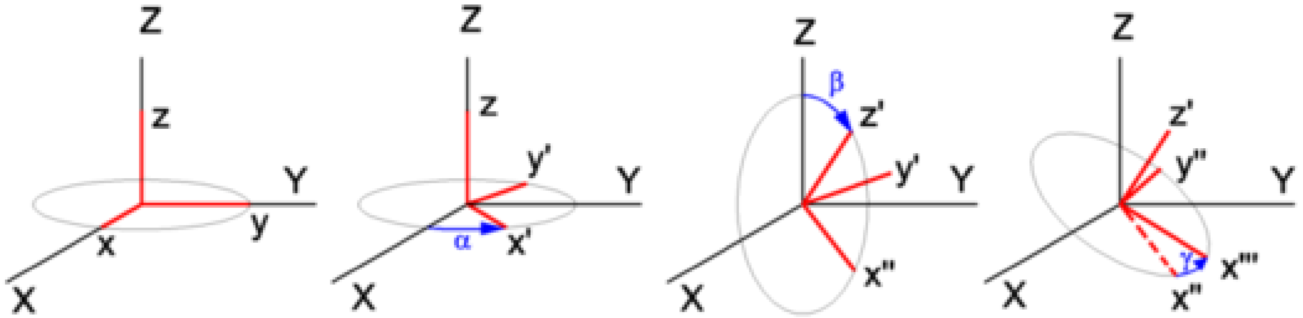
$$0.5\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}+0.5\begin{pmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}=\begin{pmatrix}0.5 & -0.5 & 0\\ 0.5 & 0.5 & 0\\ 0 & 0 & 1\end{pmatrix}$$

- Result isn't a rotation matrix!
- Need Gram-Schmidt orthonormalization
- → Interpolation computationally costly with rotation matrices

Gimbal Lock



Euler Angles



Euler Angles

- Concatenation (yes, with matrices – somewhat difficult)
- Rotation (Gimbal lock problem)
- Interpolation (no, with matrices)

Euler Angles

- Concatenation (yes, with matrices – somewhat difficult)
- Rotation (Gimbal lock problem)
- Interpolation (no, with matrices)

ϕ →

ψ

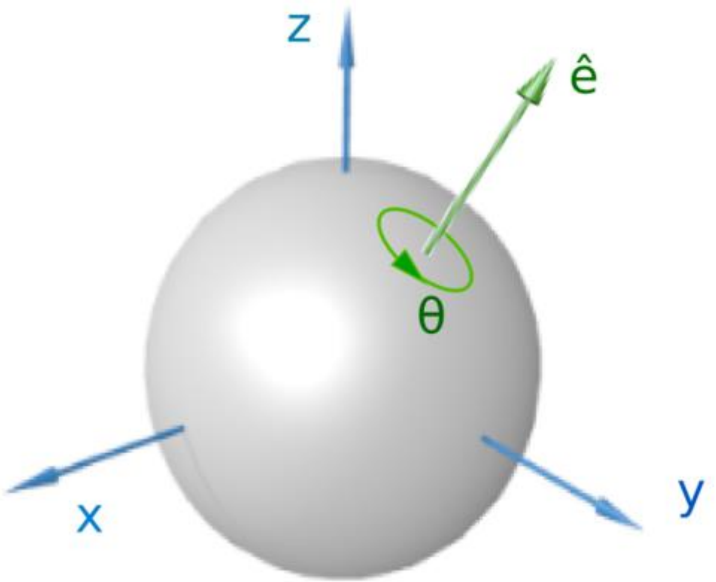
θ

$$\mathbf{R} = \mathbf{R}_Z(\psi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\phi)$$
$$= \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$
$$= \begin{pmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix}$$

$$\phi = \mathrm{Atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right)$$
$$\psi = -\mathrm{Atan2}\left(\frac{r_{21}}{\cos(\phi)}, \frac{r_{11}}{\cos(\phi)}\right)$$
$$\theta = \mathrm{Atan2}\left(\frac{r_{32}}{\cos(\phi)}, \frac{r_{33}}{\cos(\phi)}\right)$$

Euler’s Rotation Theorem

- Every 3D rotation can be represented by an axis and an angle.

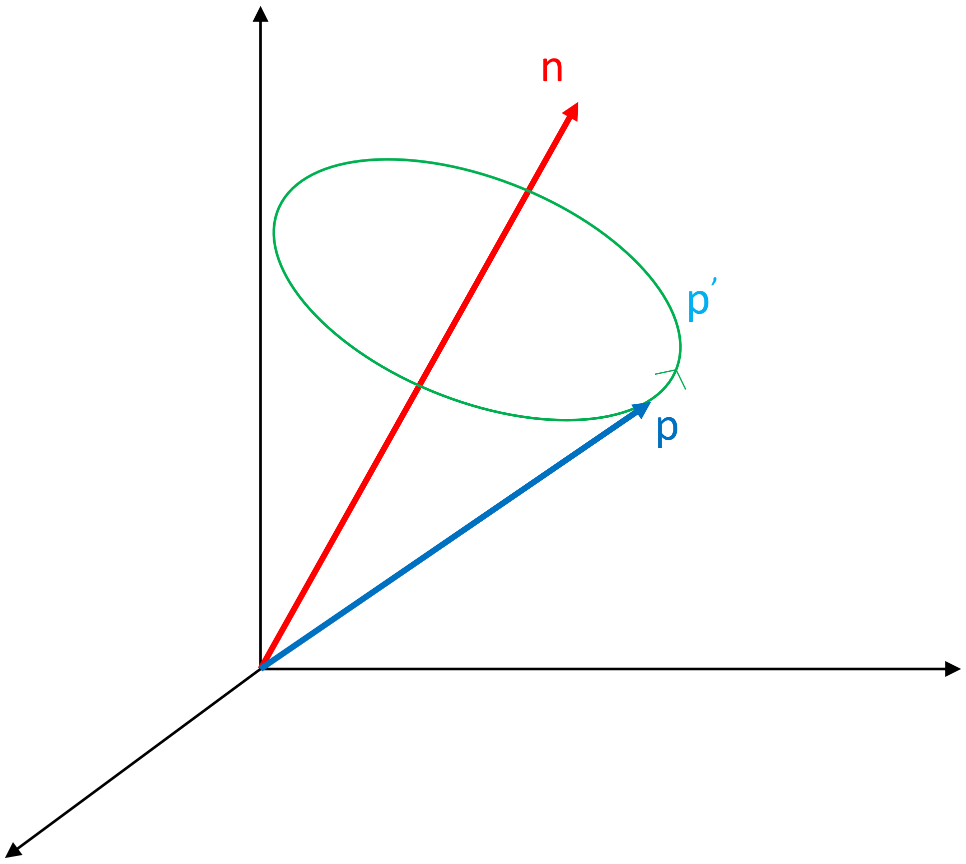


Rodrigues Rotation Theorem

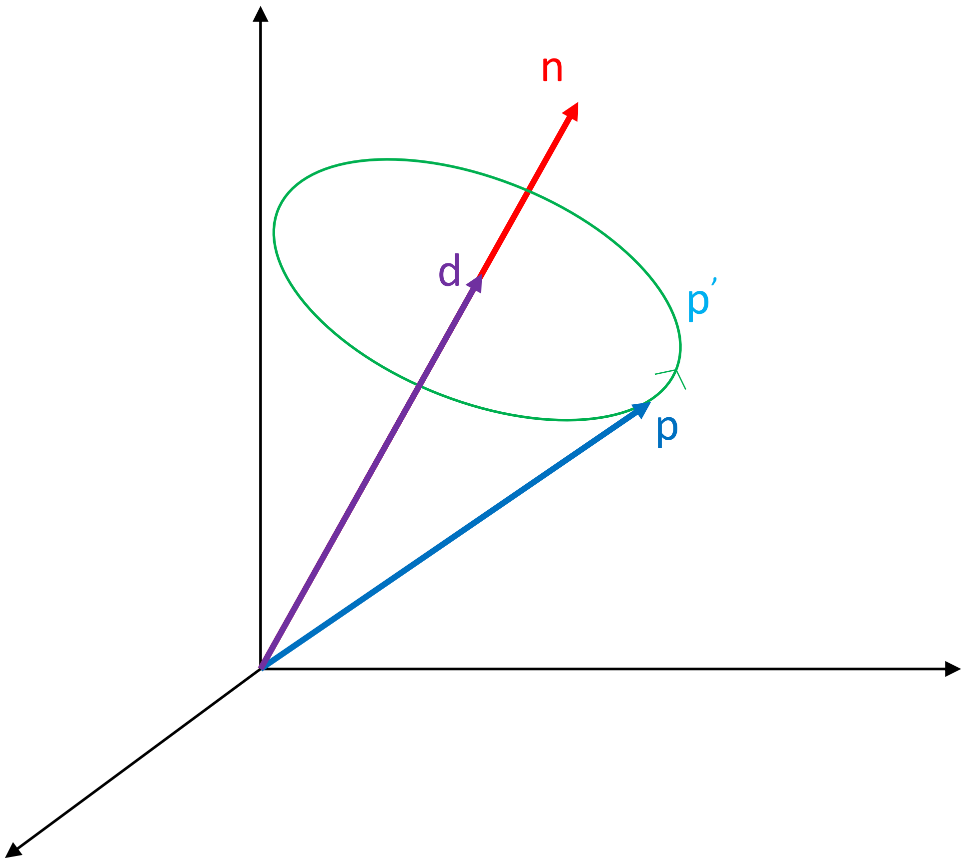
- Given axis $\hat{r}$, angle θ and point p rotation is

$$\mathrm{R}(\hat{\mathbf{r}}, \theta, \mathbf{p}) = \mathbf{p}\cos\theta + (\hat{\mathbf{r}} \times \mathbf{p})\sin\theta + \hat{\mathbf{r}}(\hat{\mathbf{r}} \bullet \mathbf{p})(1 - \cos\theta)$$

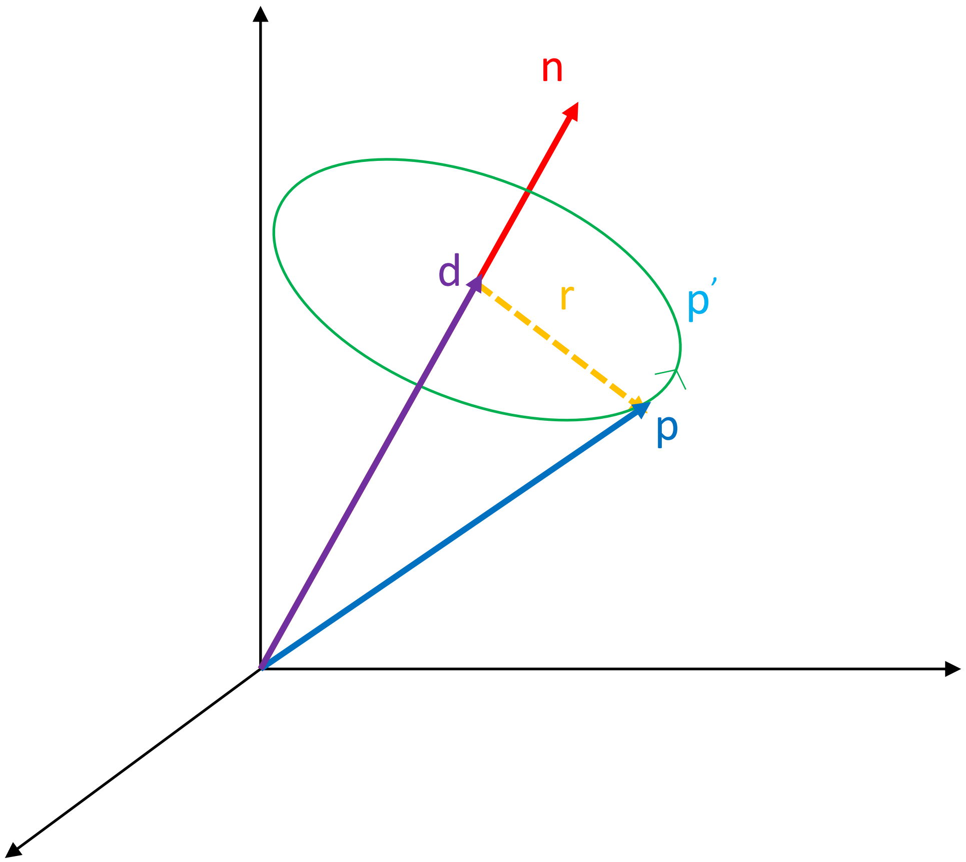
Rodrigues Formula



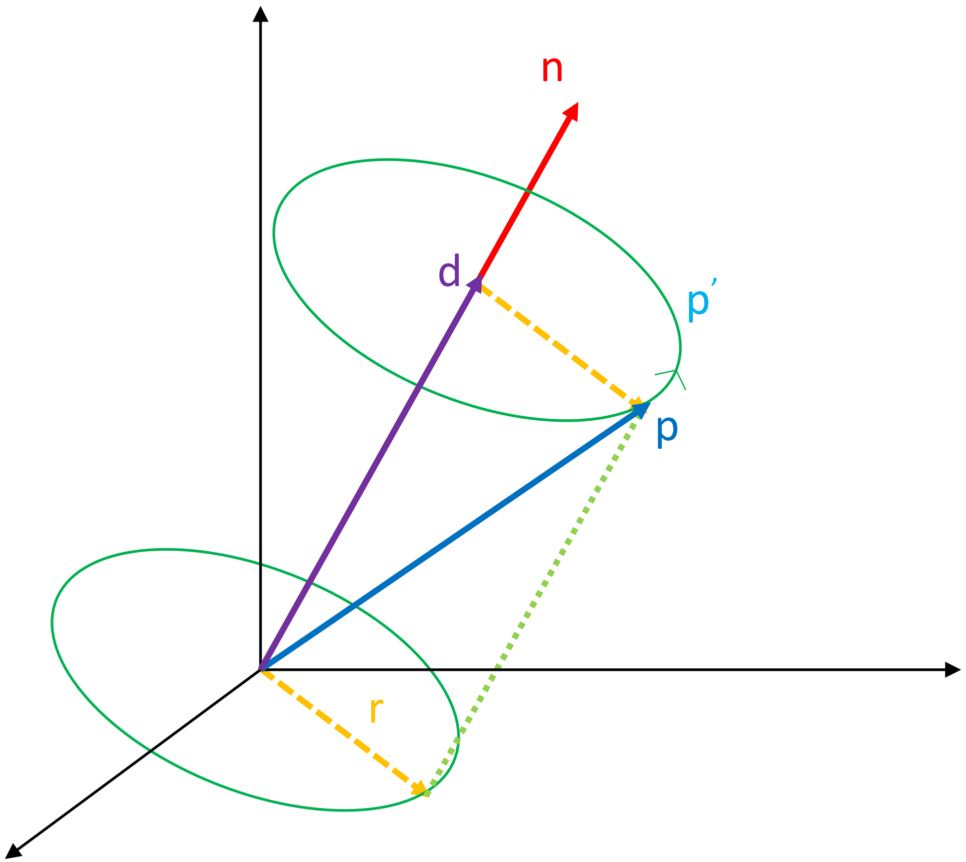
Rodrigues Formula



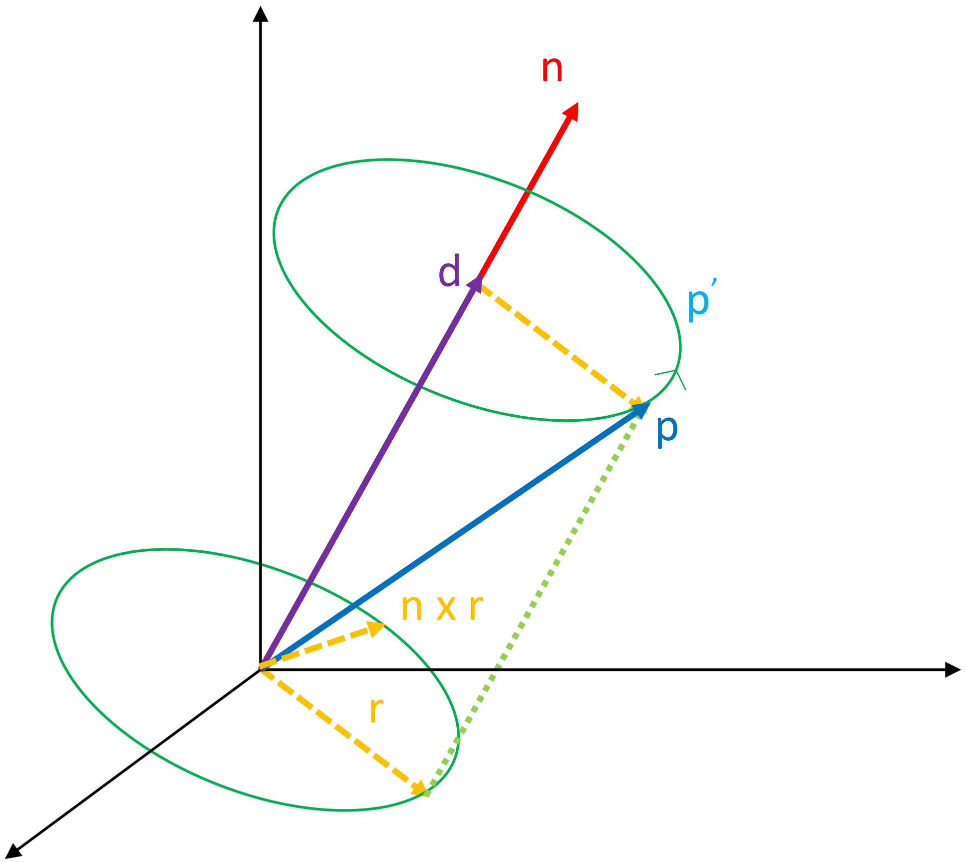
Rodrigues Formula



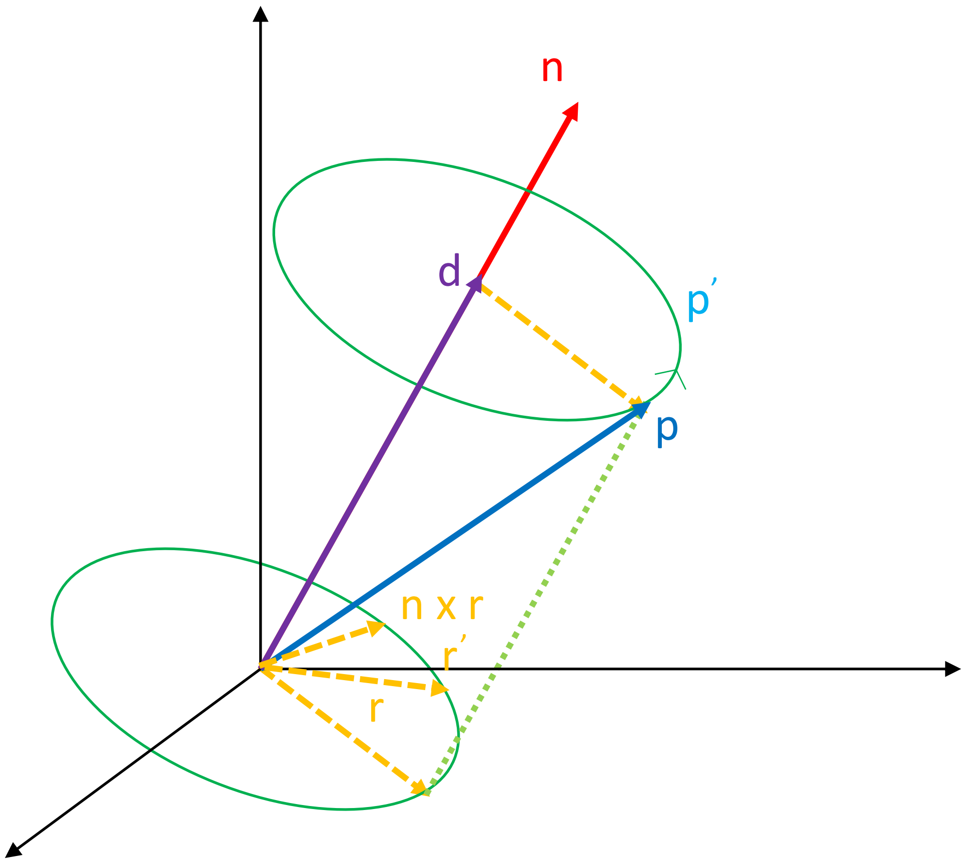
Rodrigues Formula



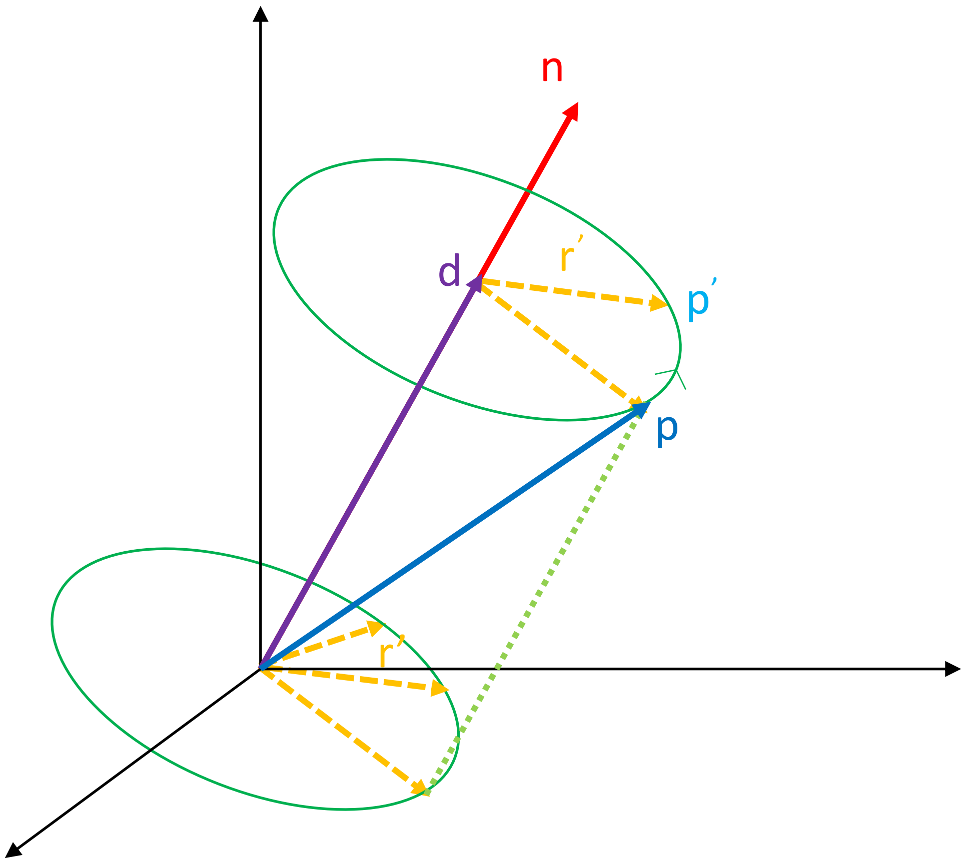
Rodrigues Formula



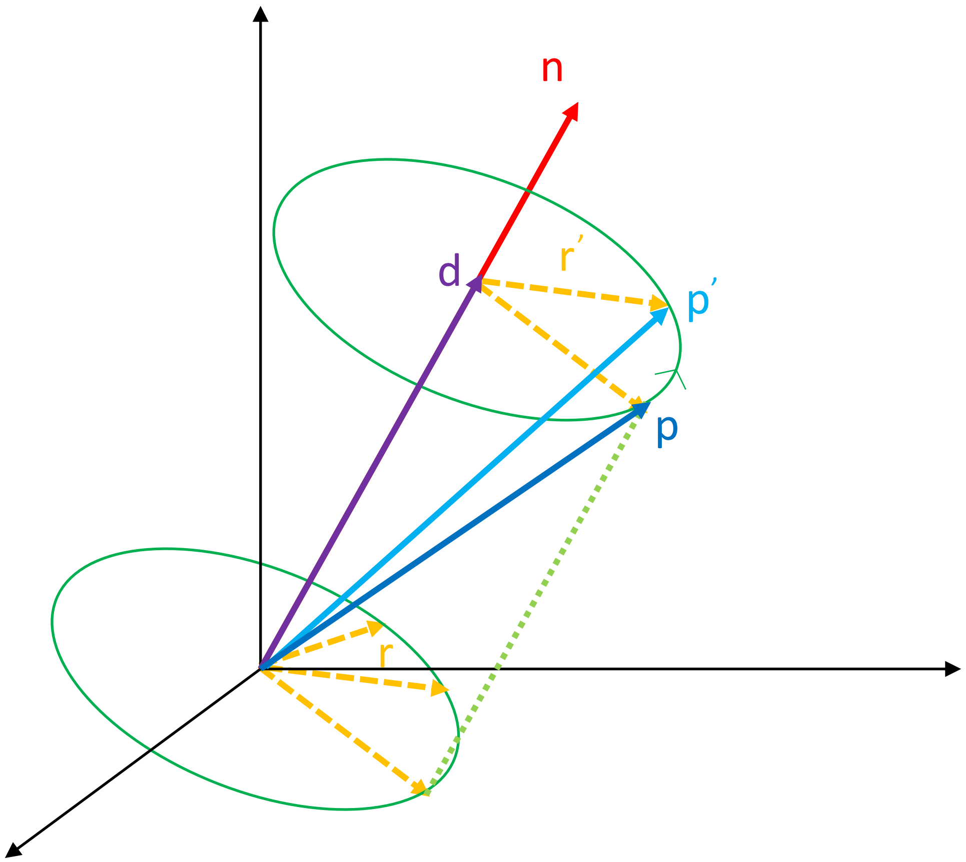
Rodrigues Formula



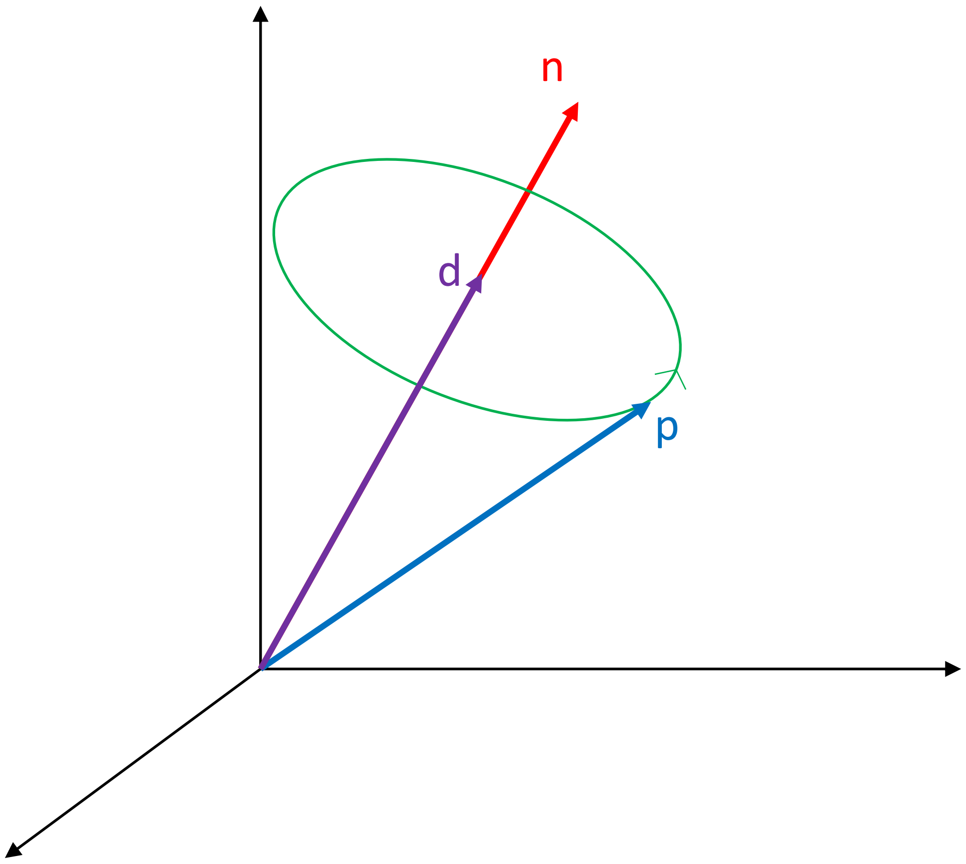
Rodrigues Formula



Rodrigues Formula

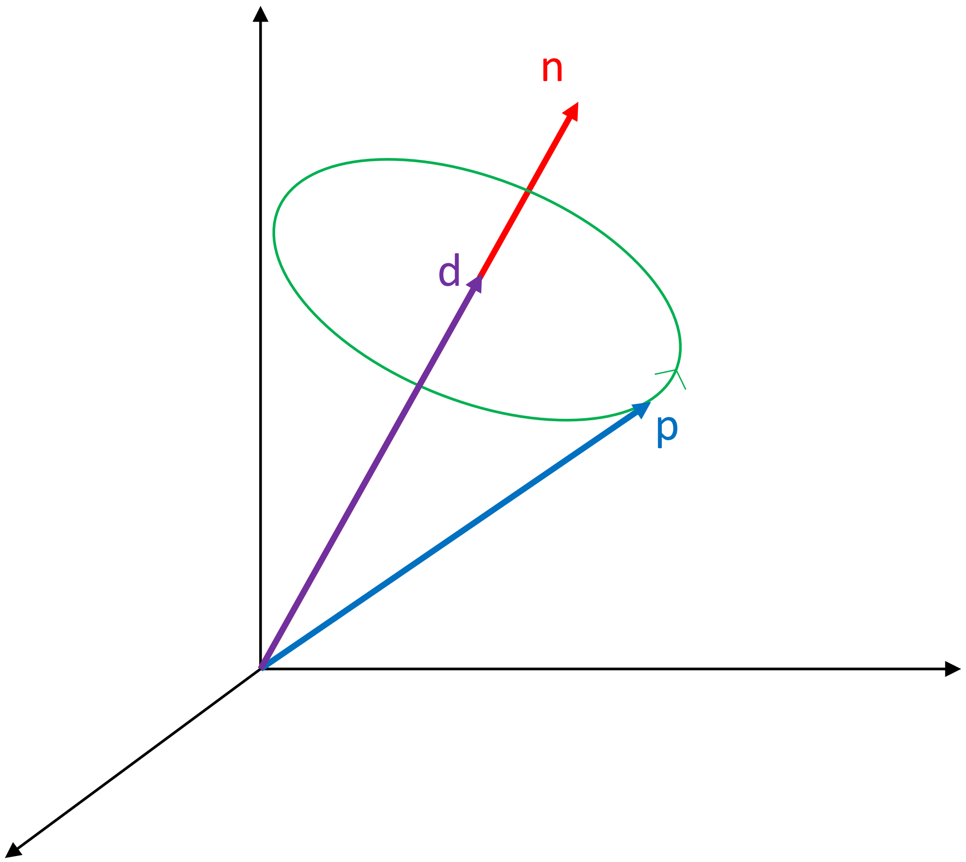


Rodrigues Formula



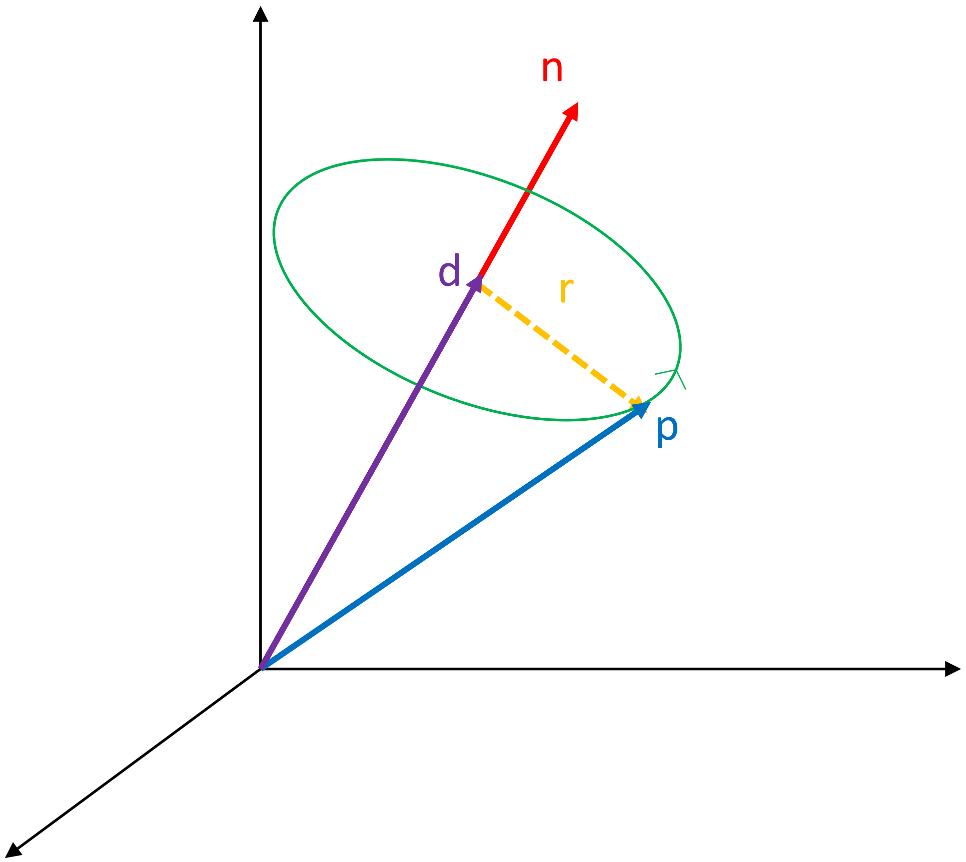
Rodrigues Formula

$$d = (n \cdot p) * n$$



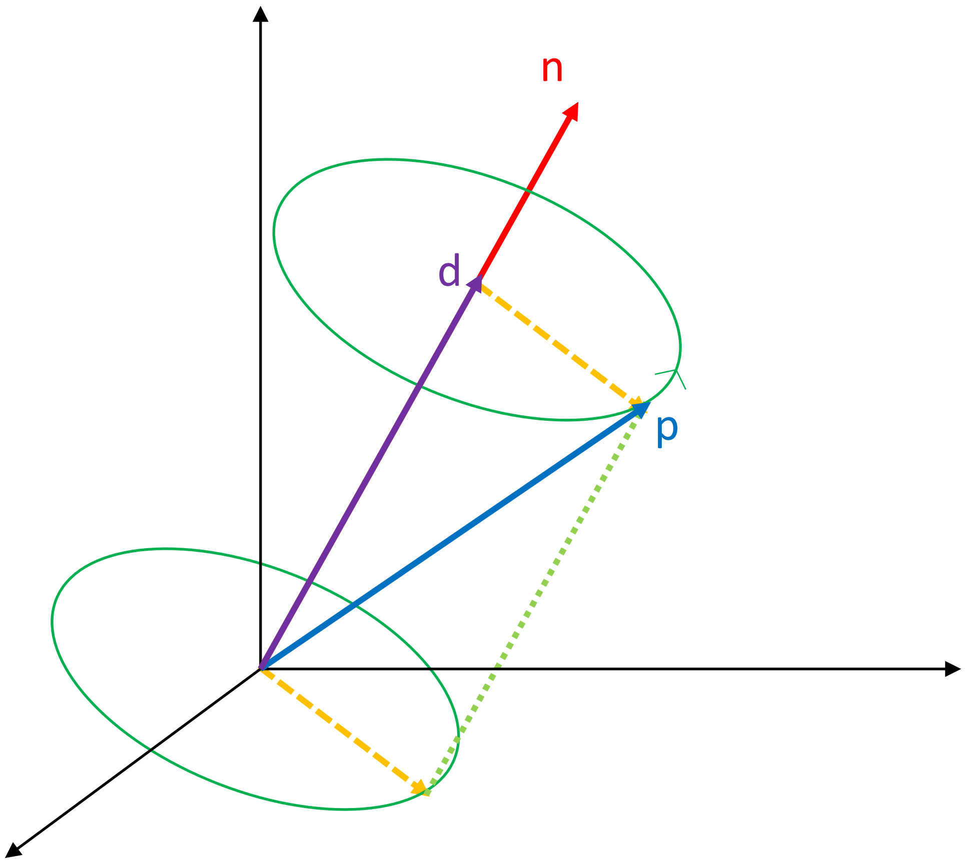
Rodrigues Formula

$$d = (n \cdot p) * n$$



Rodrigues Formula

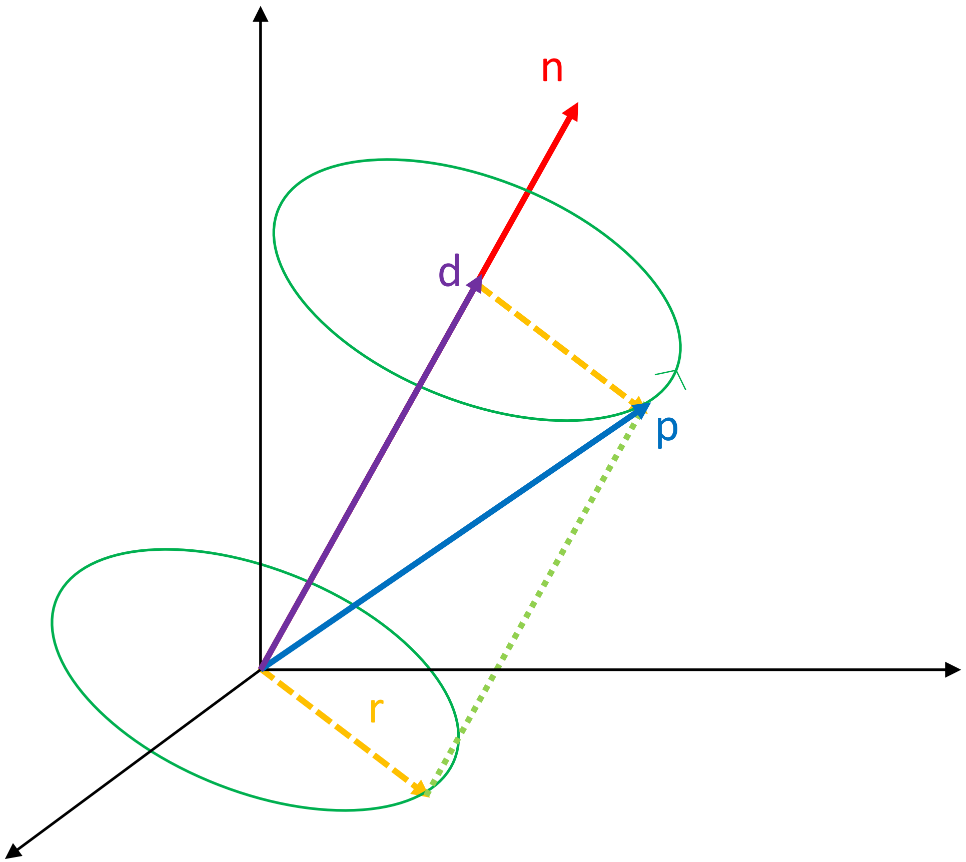
$$d = (n \cdot p) * n$$



Rodrigues Formula

$$d = (n \cdot p) * n$$

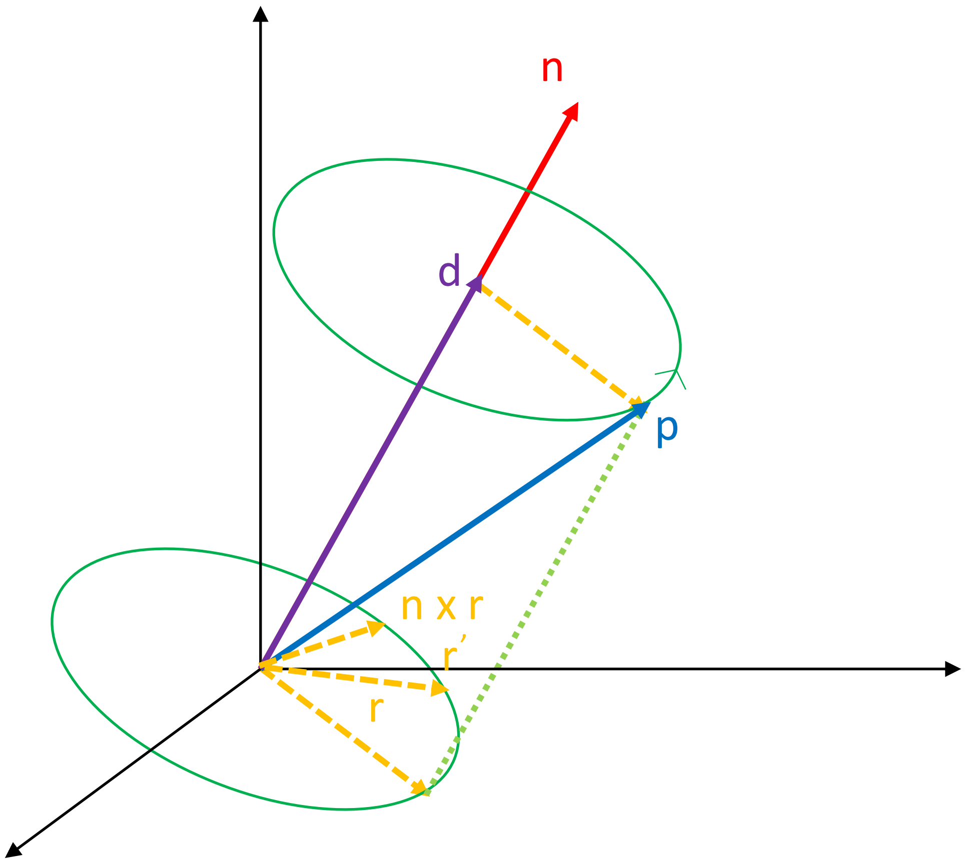
$$r = p - d$$



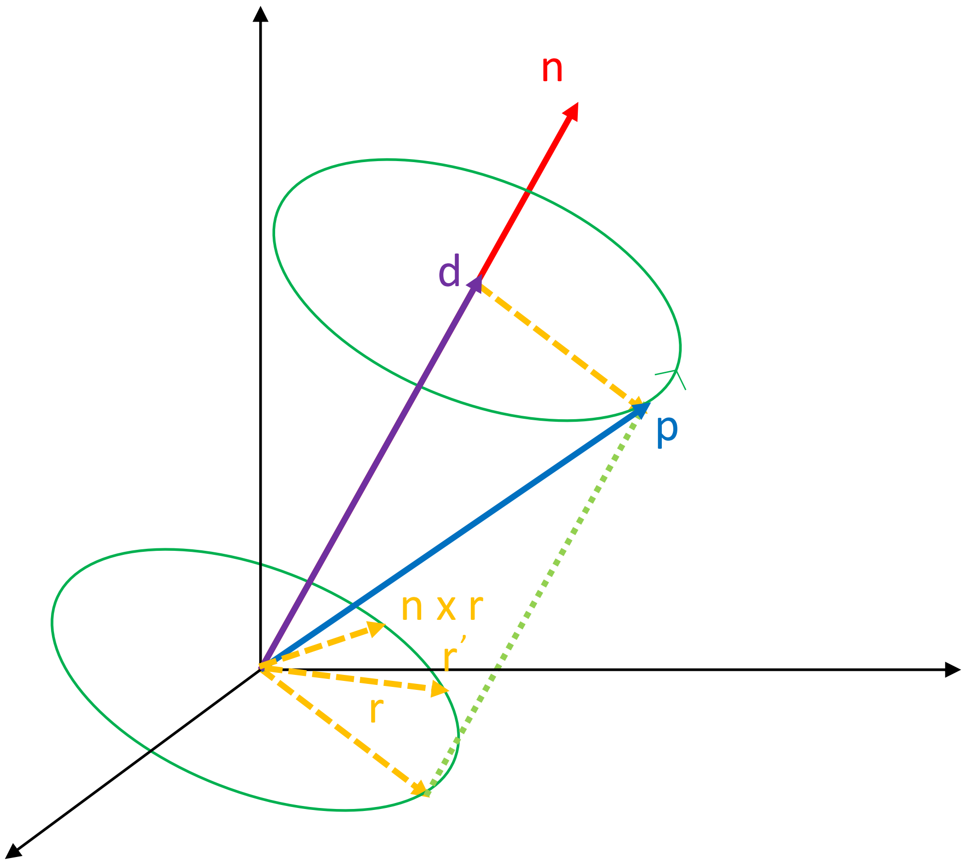
Rodrigues Formula

$$d = (n \cdot p) * n$$

$$r = p - d$$



Rodrigues Formula

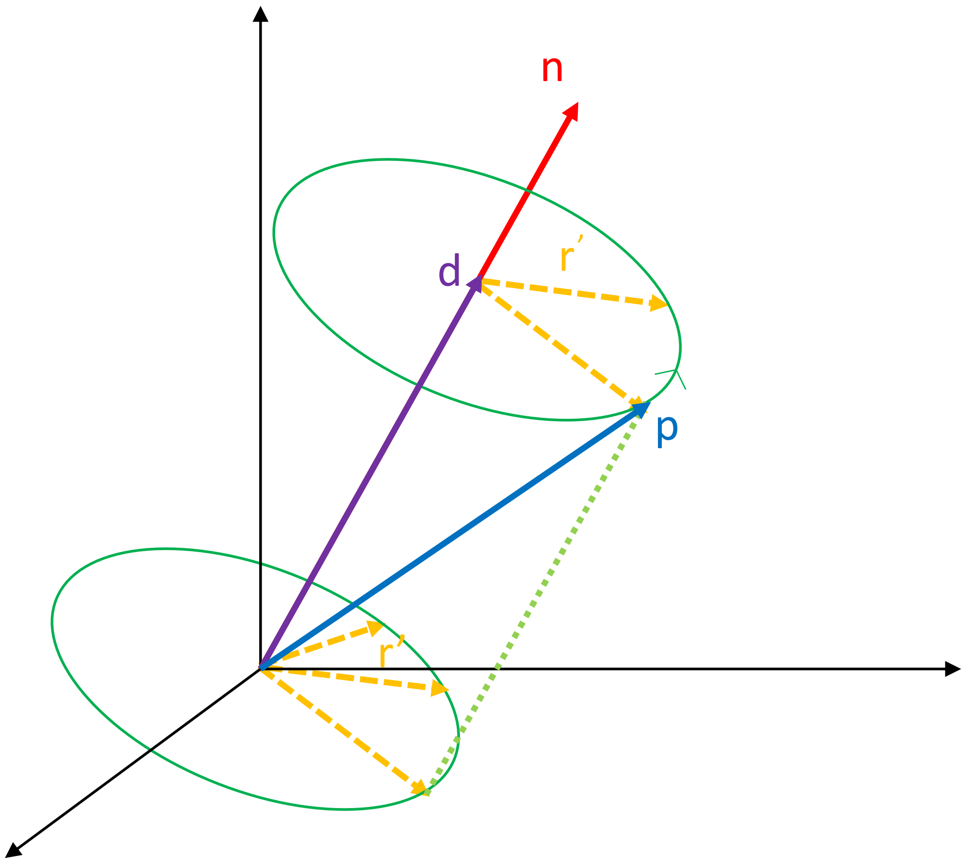


$$d = (n \cdot p) * n$$

$$r = p - d$$

$$r' = r cos\theta + (n \times r) sin\theta$$

Rodrigues Formula

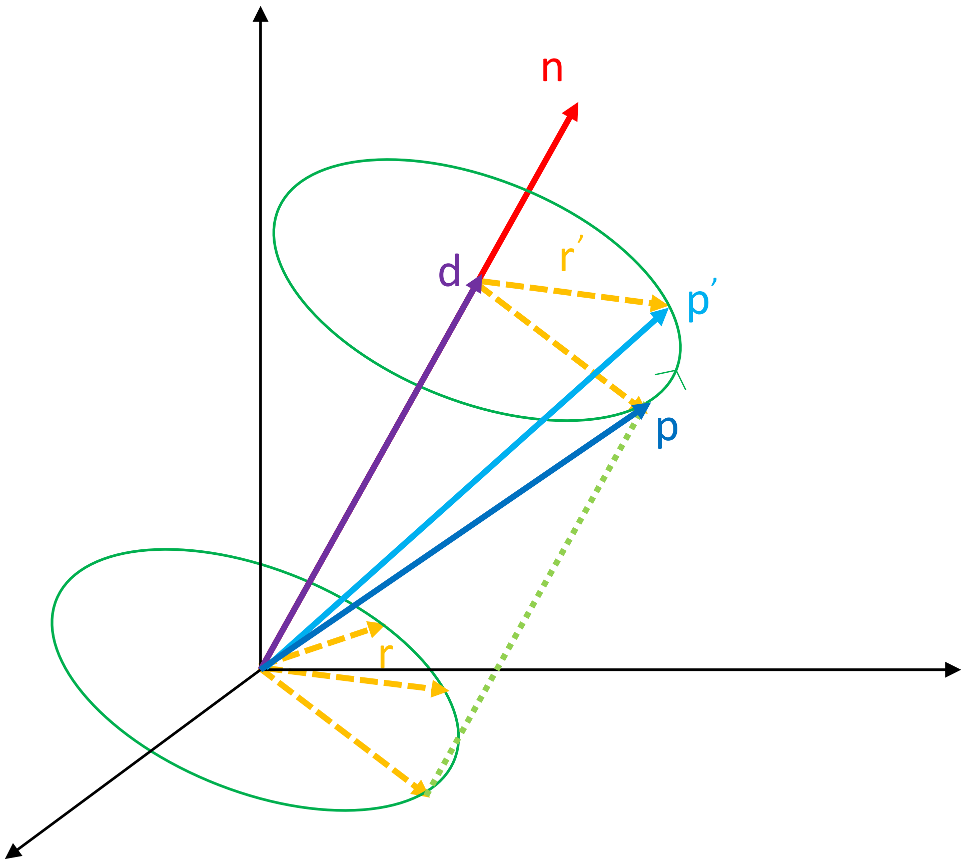


$$d = (n \cdot p) * n$$

$$r = p - d$$

$$r' = r cos\theta + (n \times r) sin\theta$$

Rodrigues Formula



$$d = (n \cdot p) * n$$

$$r = p - d$$

$$r' = r cos\theta + (n \times r) sin\theta$$

$$\mathrm{p}' = d + r'$$

Euler-Rodrigues Parameters

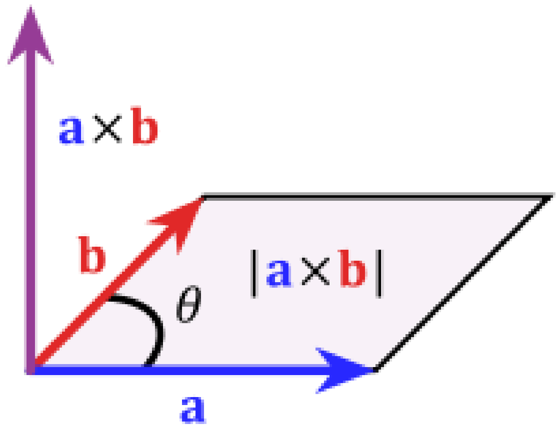
- Given $a^2 + b^2 + c^2 + d^2 = 1$ we can build a rotation
- Set
  - $a = \cos(\theta/2)$
  - $r = (b, c, d) = \sin(\theta/2)n$
- Then rotate using

$$\mathbf{R(a, r, p) = 2a(r \times p) + 2(r \times (r \times p)) + p}$$

(also has a matrix representation)

Axis-Angle Rotation

- Concatenation (yes, with matrices – somewhat difficult)
- Rotation (no Gimbal lock problem)
- Interpolation (find axis of rotation by taking cross product of 2 axes)

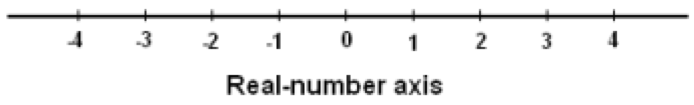


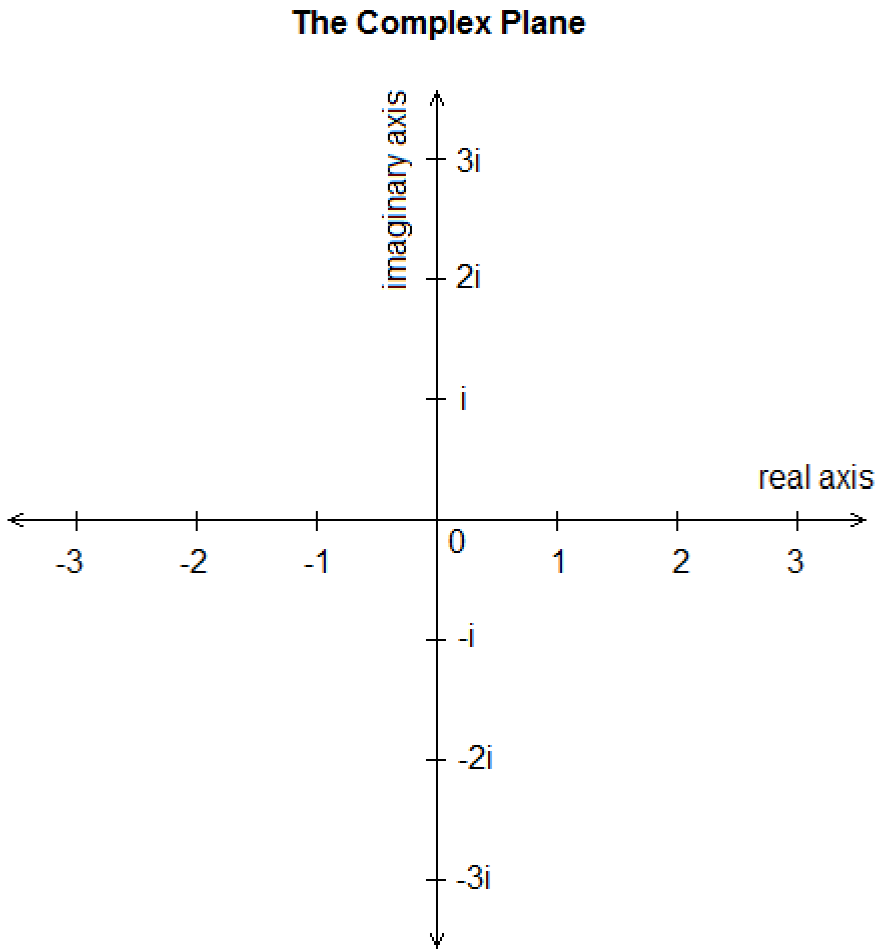
Complex Numbers

- 2D numbers: $a + bi$
- Where $i = \sqrt{-1}$

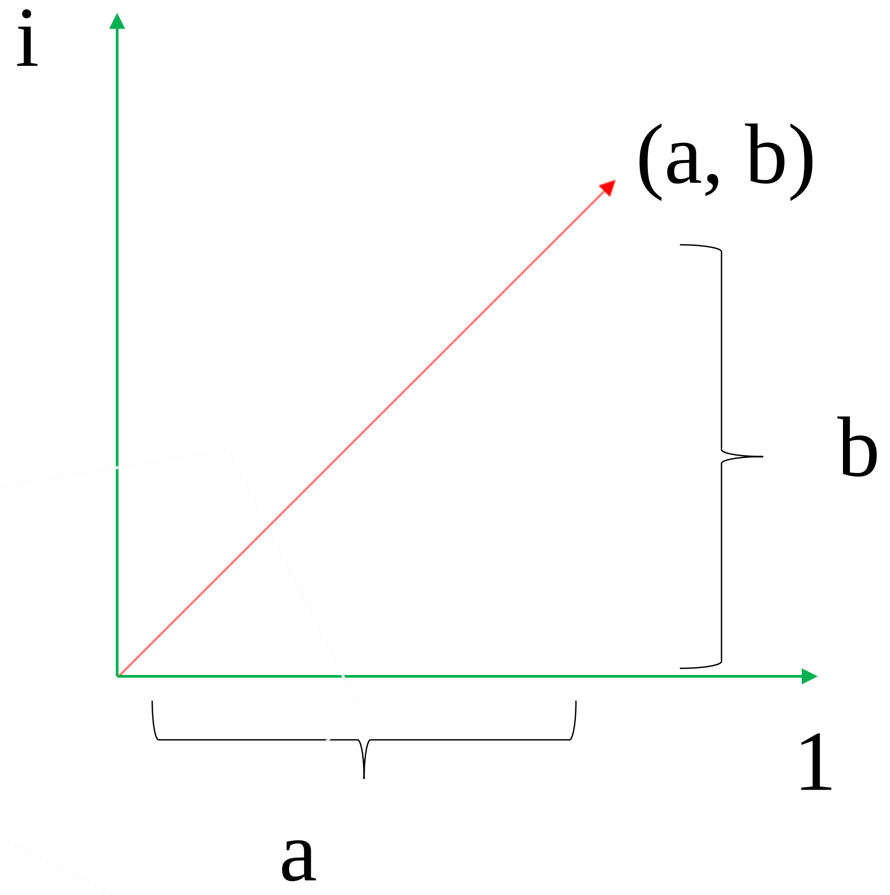
Complex Numbers

- 2D numbers: $a + bi$
- Where $i = \sqrt{-1}$

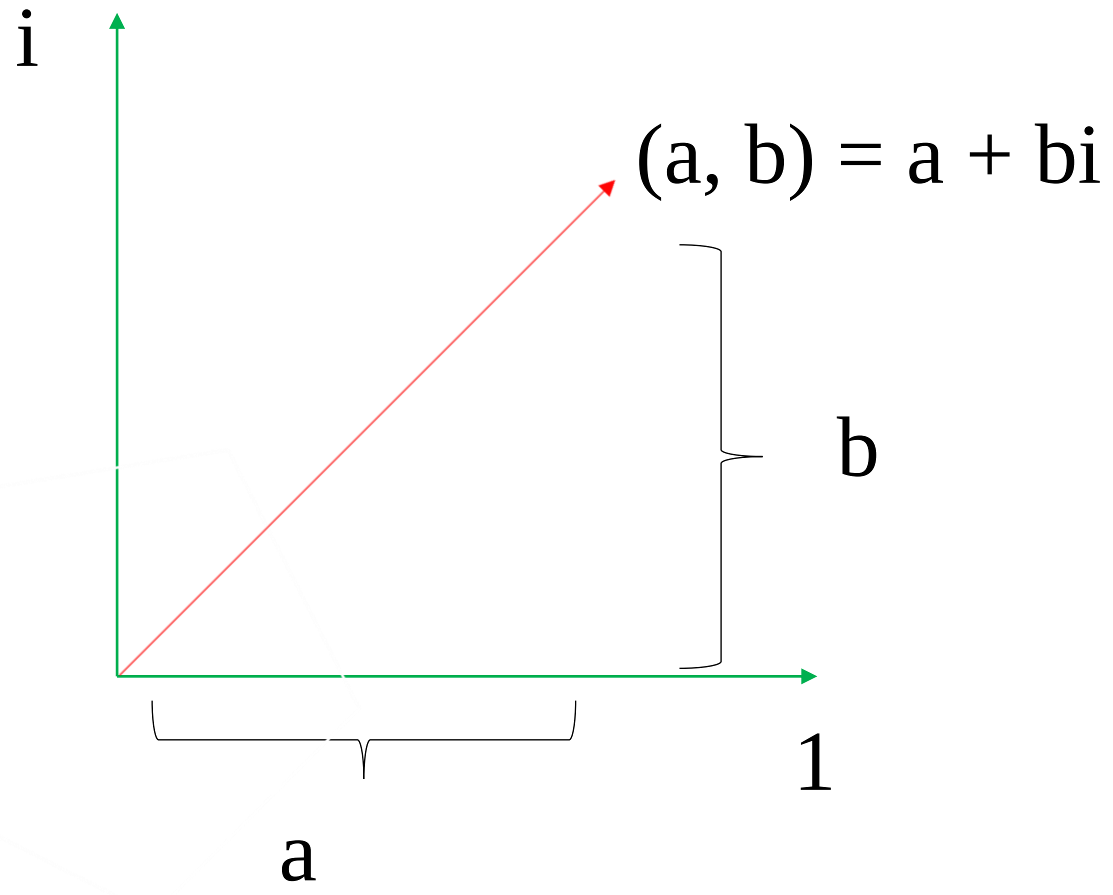




Complex Numbers



Complex Numbers

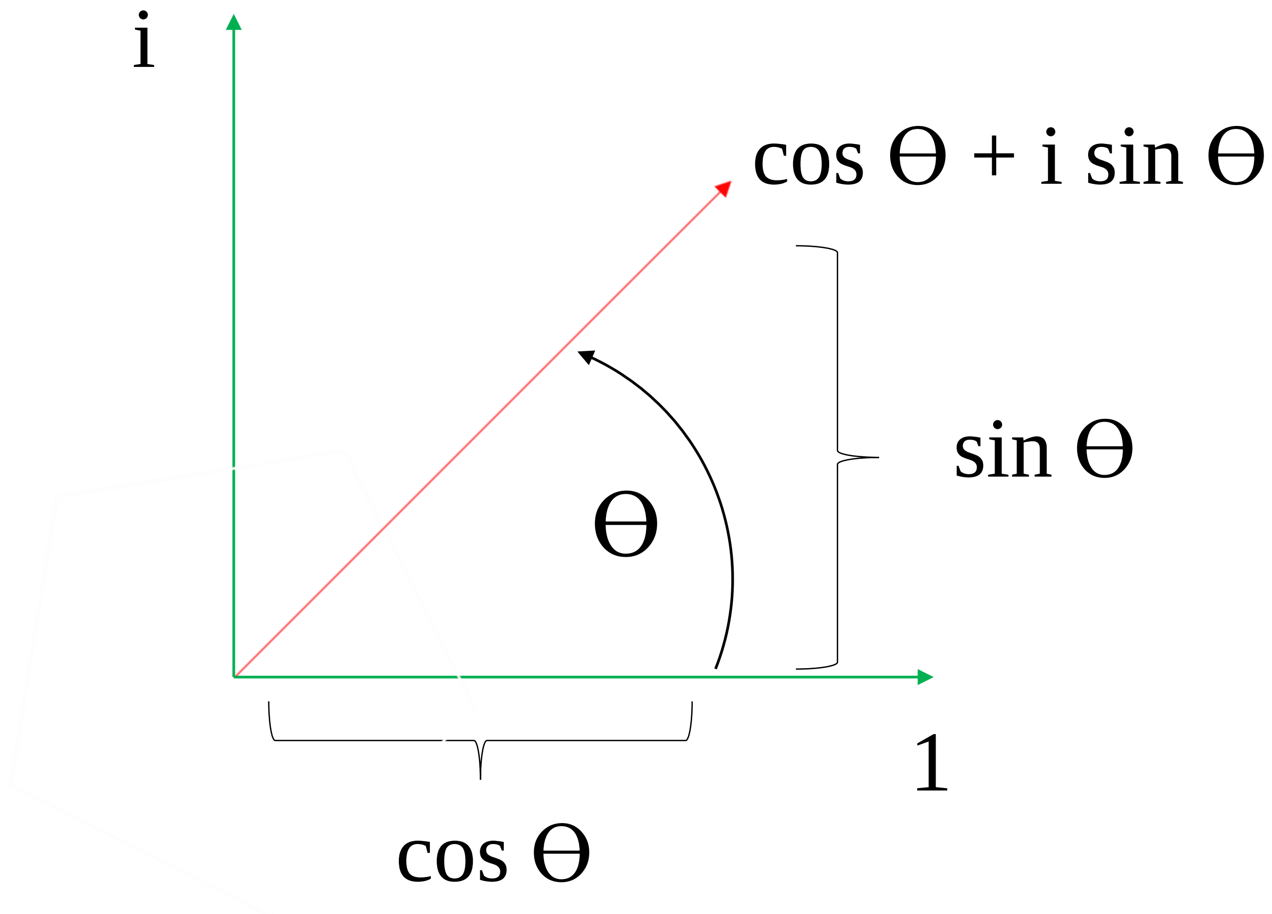


Complex Numbers

- Multiplication:
- $(a + bi)(c + di) = (ac - bd) + (bc + ad)i$

- $(a, b)(c, d) = (ac - bd, bc + ad)$

Complex Numbers: Unit Length



Complex Numbers

- Multiply general complex number by unit complex number:

- $(x + yi)(cos\theta + isin\theta) = (xcos\theta - ysin\theta) + i(xsin\theta + ycos\theta)$

general unit

Complex Numbers

- Multiply general complex number by unit complex number:

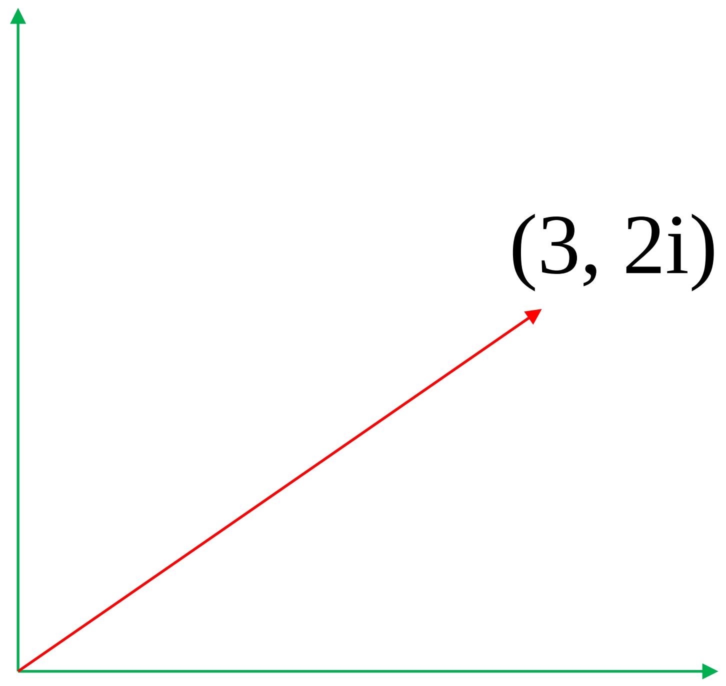
- $(x + yi)(cos\theta + isin\theta) = (xcos\theta - ysin\theta) + i(xsin\theta + ycos\theta)$

general

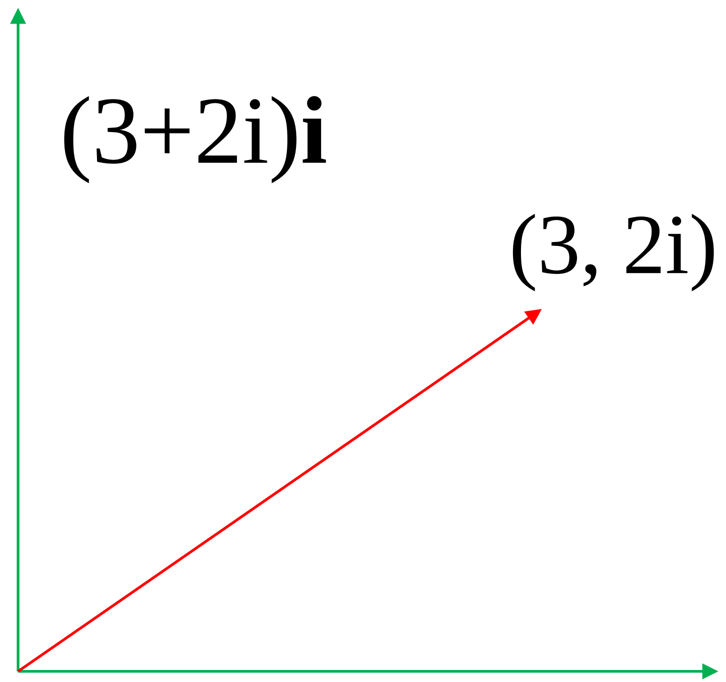
unit

2D rotation!

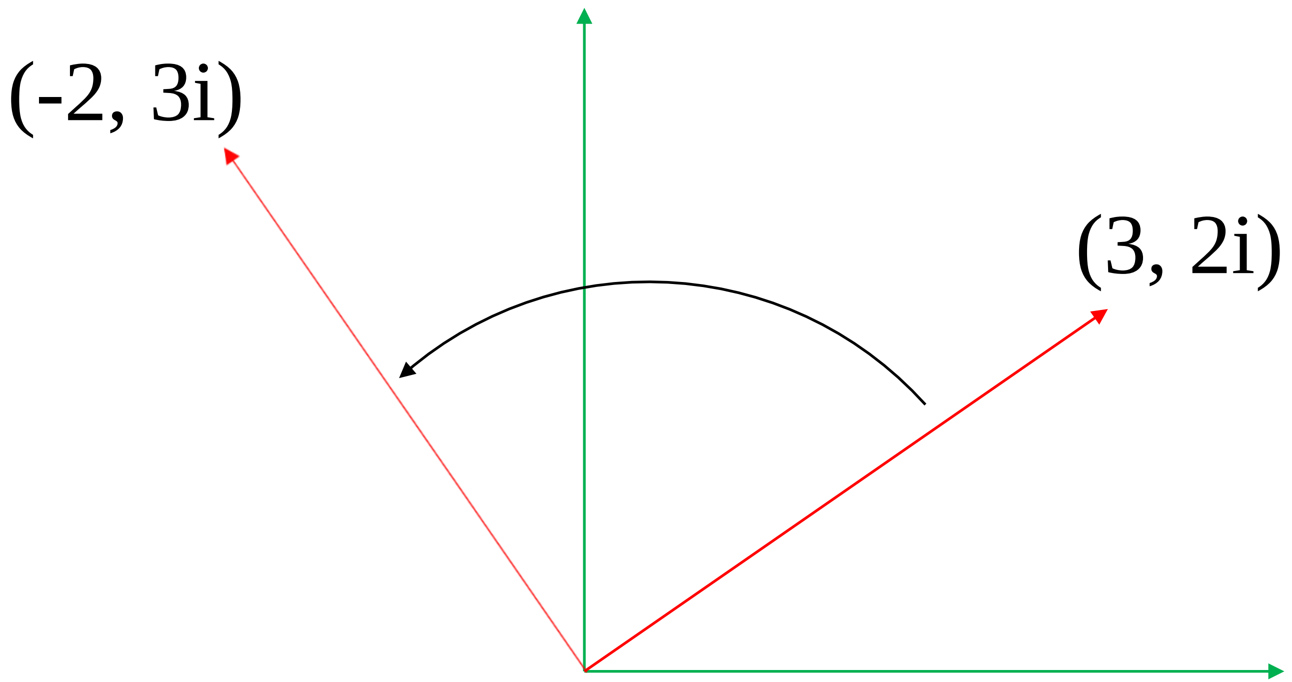
Example Complex Number Rotation by i



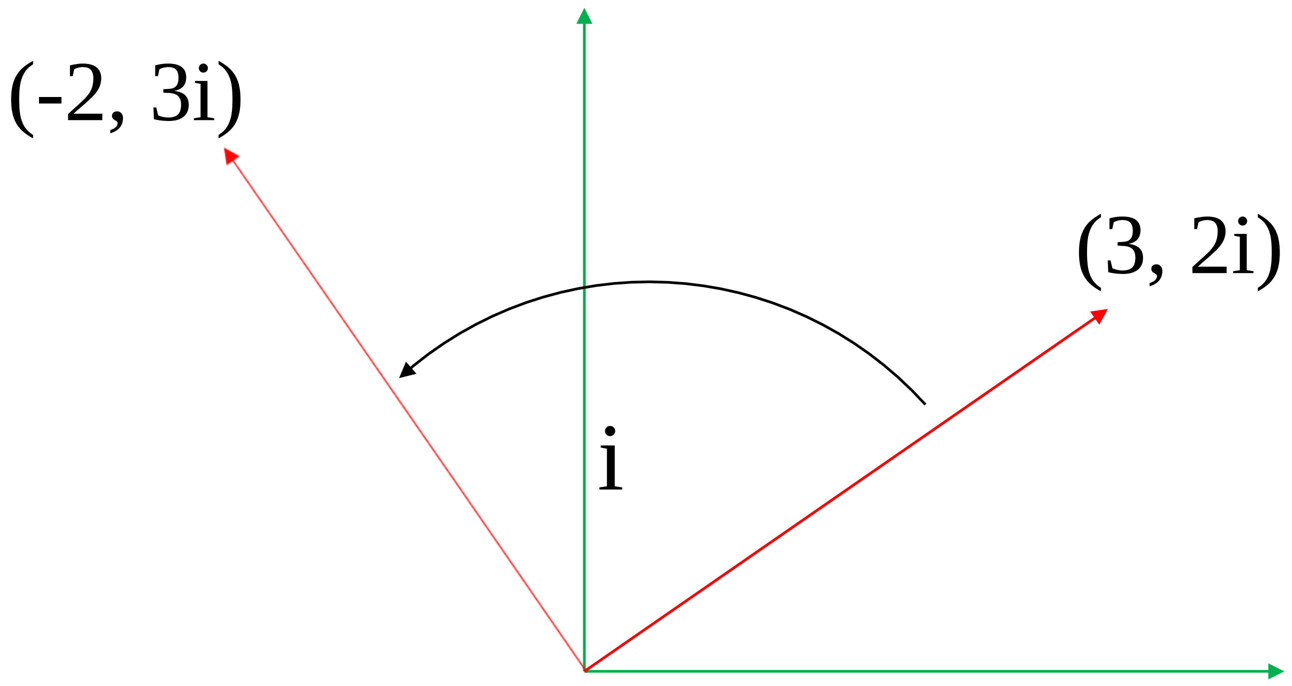
Example Complex Number Rotation by i



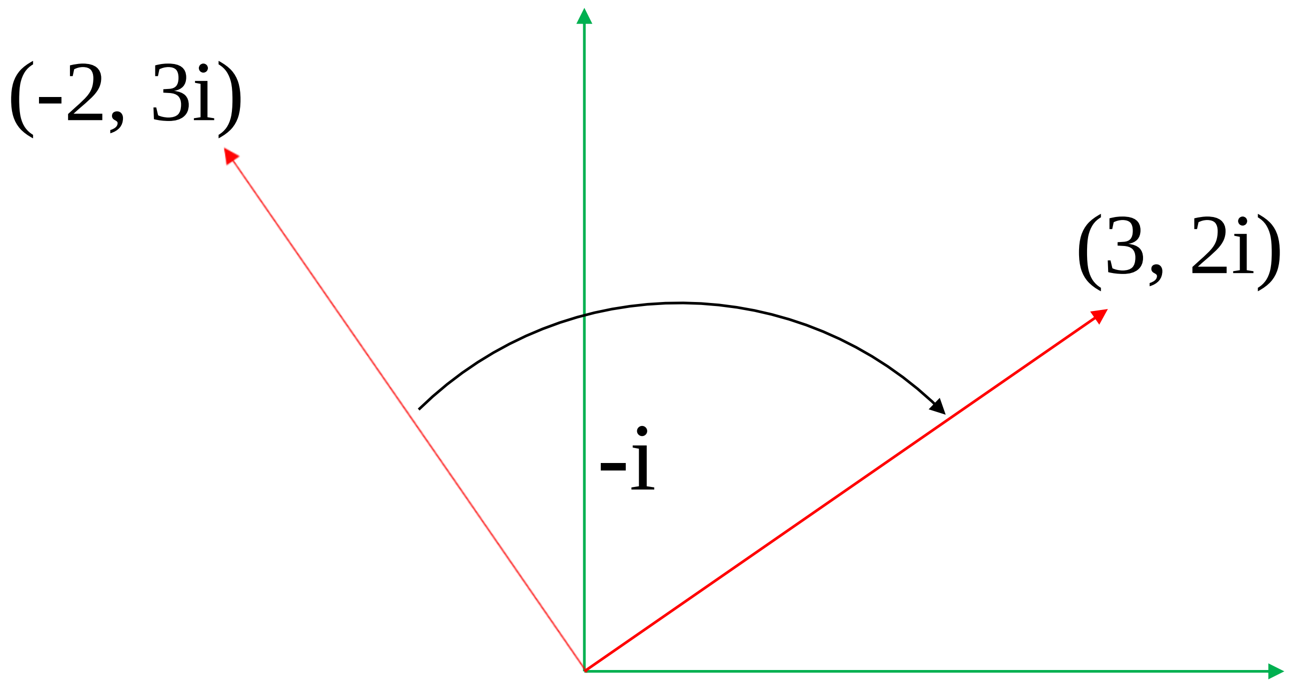
Example Complex Number Rotation by i



Unit Complex Numbers are Rotations!



Inverse Rotation



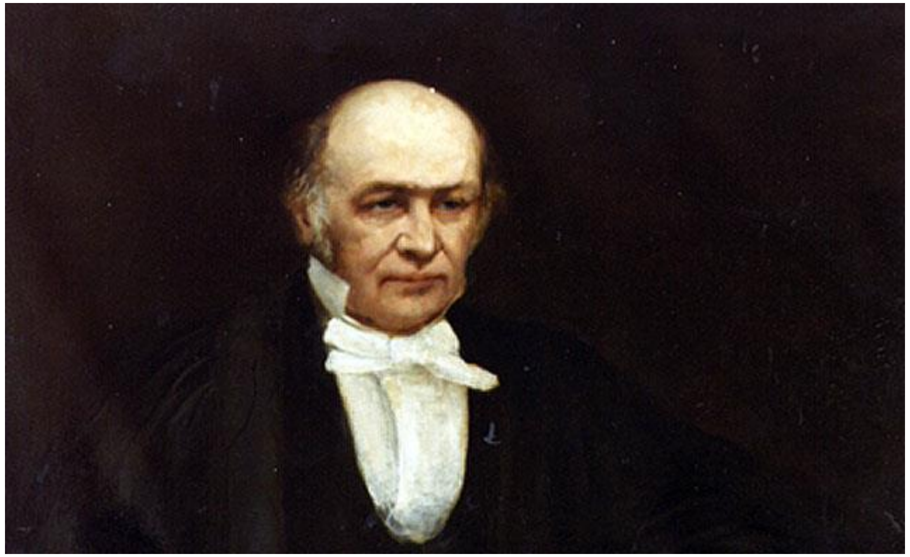
Computational Efficiency?

- Rotation matrix vs. complex number multiplication

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} \qquad (p_x, p_y)(u_x, u_y)$$

3D Complex Numbers?

$$(a_0 + b_0 i + c_0 j)$$



3D Complex Numbers?

- Problem with multiplication

$$(a_0 + b_0 i + c_0 j)(a_1 + b_1 i + c_1 j)$$

3D Complex Numbers?

- Problem with multiplication

$$
\begin{aligned}
(a_0 + b_0 i + c_0 j)(a_1 + b_1 i + c_1 j) = \quad & (a_0 a_1 - b_0 b_1 - c_0 c_1) \\
+ \quad & (a_0 b_1 + b_0 a_1) i \\
+ \quad & (a_0 c_1 + c_0 a_1) j \\
+ \quad & b_0 c_1 ij + c_0 b_1 ji
\end{aligned}
$$

3D Complex Numbers?

- Problem with multiplication

$$
\begin{aligned}
(a_0 + b_0 i + c_0 j)(a_1 + b_1 i + c_1 j) = \quad & (a_0 a_1 - b_0 b_1 - c_0 c_1) \\
+ \quad & (a_0 b_1 + b_0 a_1) i \\
+ \quad & (a_0 c_1 + c_0 a_1) j \\
+ \quad & b_0 c_1 ij + c_0 b_1 ji
\end{aligned}
$$

Not a division algebra → not every non-zero number has an inverse!

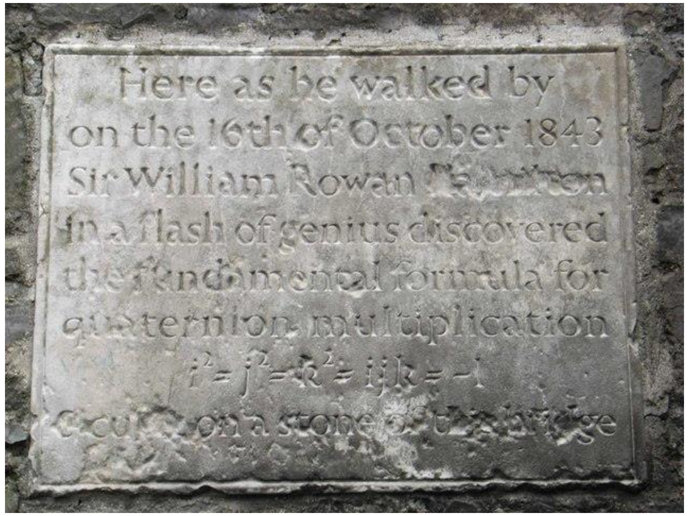
Quaternions

- Hamilton's insight: multiplication possible with 3 imaginary values

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

$$ij = k \qquad jk = i \qquad ki = j$$

$$ji = -k \qquad kj = -i \qquad ik = -j$$



Quaternions

- Are extended complex numbers in 4D

$$a + bi \rightarrow w + xi + yj + zk$$

$$(w, x, y, z)$$

$$(w, v)$$

Quaternions extend Complex Numbers

- Suppose y = z = 0 then,

$$a + bi \rightarrow w + xi + 0j + 0k = w + xi$$

Quaternion Multiplication

- Provides concatenation of rotations
- Take $\mathbf{q}_0 = (w_0, \mathbf{v}_0)$ $\mathbf{q}_1 = (w_1, \mathbf{v}_1)$

$$\mathbf{q}_1\mathbf{q}_0 = (w_1 w_0 - \mathbf{v}_1 \bullet \mathbf{v}_0, w_1\mathbf{v}_0 + w_0\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_0)$$

- If w_0, w_1 are zero:

$$\mathbf{q}_1\mathbf{q}_0 = (-\mathbf{v}_1 \bullet \mathbf{v}_0, \mathbf{v}_1 \times \mathbf{v}_0)$$

- Non-commutative:

$$\mathbf{q}_1\mathbf{q}_0 \neq \mathbf{q}_0\mathbf{q}_1$$

Quaternions

- Identity quaternion is $(1, 0, 0, 0)$
  - applies no rotation
  - remains at reference orientation
- $\mathbf{q}^{-1}$ is inverse
  - $\mathbf{q} \cdot \mathbf{q}^{-1}$ gives identity quaternion

Quaternion Rotation

- Like complex numbers, unit quaternions represent rotations
- For 3D rotation:

$$w = \cos(\theta/2)$$

$$(x, y, z) = v = \sin(\theta/2)n$$

Quaternion Rotation

- We can easily obtain the **inverse quaternion** for any unit (normalized) quaternion

$$q = (w, v) = (\cos(\theta/2), \sin(\theta/2)n)$$

$$q^{-1} = (w, v)^{-1} = (\cos(-\theta/2), \sin(-\theta/2)n)$$

Quaternion Rotation

- We can easily obtain the **inverse quaternion** for any unit (normalized) quaternion

$$q = (w, v) = (\cos(\theta/2), \sin(\theta/2)n)$$

$$q^{-1} = (w, v)^{-1} = (\cos(-\theta/2), \sin(-\theta/2)n)$$

$$(w, v)^{-1} = (\cos(\theta/2), -\sin(\theta/2)n)$$

Quaternion Rotation

- We can easily obtain the **inverse quaternion** for any unit (normalized) quaternion

$$q = (w, v) = (\cos(\theta/2), \sin(\theta/2)n)$$

$$q^{-1} = (w, v)^{-1} = (\cos(-\theta/2), \sin(-\theta/2)n)$$

$$(w, v)^{-1} = (\cos(\theta/2), -\sin(\theta/2)n)$$

$$(w, v)^{-1} = (w, -v)$$

Quaternion Rotation

- Have vector **p**, unit quaternion *q*
- Treat **p** as quaternion *p* = (0, **p**)
- 3D rotation of *p* by *q* is

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

Quaternion Rotation

- Have vector **p**, unit quaternion *q*
- Treat **p** as quaternion *p* = (0, **p**)
- 3D rotation of *p* by *q* is

$$\mathbf{p}' = \mathbf{q}\,\mathbf{p}\,\mathbf{q}^{-1}$$

$$p + 2w(v \times p) + 2(v \times (v \times p))$$

Quaternion Rotation

- **Quaternion Rotation**
- **Euler-Rodrigues**

- $w = \cos(\theta/2)$
- $a = \cos(\theta/2)$

Quaternion Rotation

- **Quaternion Rotation**

- $w = \cos(\theta/2)$
- $(x, y, z) = v = \sin(\theta/2)n$

- **Euler-Rodrigues**

- $a = \cos(\theta/2)$
- $(b, c, d) = r = \sin(\theta/2)n$

Quaternion Rotation

- **Quaternion Rotation**

- $w = \cos(\theta/2)$
- $(x, y, z) = v = \sin(\theta/2)n$

- $p + 2w(v \times p) + 2(v \times (v \times p))$

- **Euler-Rodrigues**

- $a = \cos(\theta/2)$
- $(b, c, d) = r = \sin(\theta/2)n$

- $p + 2a(r \times p) + 2(r \times (r \times p))$

Matrix Form

- Decompose 2D rotation matrix into:

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

- Set

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{J} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

- Then

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \cos\theta\mathbf{I} + \sin\theta\mathbf{J}$$

Matrix Form

- **I** and **J** follow same rules as 1 and i

$$\mathbf{J}^2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} = -\mathbf{I}$$

$$(a\mathbf{I} + b\mathbf{J})(c\mathbf{I} + d\mathbf{J}) = (ac - bd)\mathbf{I} + (bc + ad)\mathbf{J}$$

- Complex numbers in another form!

Matrix Form

- For quaternions:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \quad \mathbf{Z} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Matrix Form

- Then

$$X^2 = Y^2 = Z^2 = XYZ = -I$$

- And

$$Q = wI + xX + yY + zZ$$

- Q is a quaternion!

Matrix Form

$$\mathbf{Q} = \begin{bmatrix} w & -z & y & x \\ z & w & -x & y \\ -y & x & w & z \\ -x & -y & -z & w \end{bmatrix}$$

- If $w^2 + x^2 + y^2 + z^2 = 1$
  - Unit vectors
  - Orthogonal vectors, dot products 0
  - Determinant 1
- Q is a rotation matrix!

$$\mathbf{P}' = \mathbf{QPQ}^{-1}$$

Matrix Form

- $\mathbf{Q}$ is a rotation quaternion
- Transform with half angle

$$\mathbf{P}' = \mathbf{Q}\mathbf{P}\mathbf{Q}^{-1}$$

Rotate halfway   Rotate halfway

4D Rotation

- Two types:
  - Single rotation (one plane, like 2D/3D)
  - Double rotation (two orthogonal planes)

4D Rotation

- Two types:
  - Single rotation (one plane, like 2D/3D)
  - Double rotation (two orthogonal planes)

$$(i)(w + ix + jy + kz) = -x + iw - jz + ky$$

4D Rotation

- Two types:
  - Single rotation (one plane, like 2D/3D)
  - Double rotation (two orthogonal planes)

$$(i)(w + ix + jy + kz) = -x + iw - jz + ky$$

90° rotation in (w, x) plane

4D Rotation

- Two types:
    - Single rotation (one plane, like 2D/3D)
    - Double rotation (two orthogonal planes)

$$(i)(w + ix + jy + kz) = -x + iw - jz + ky$$

90° rotation in (w, x) plane

90° rotation in (y, z) plane

4D Rotation

- Two types:
  - Single rotation (one plane, like 2D/3D)
  - Double rotation (two orthogonal planes)

$$(w + ix + jy + kz)(i) = -x + iw + jz - ky$$

90° rotation in (w, x) plane

-90° rotation in (y, z) plane

4D Rotation

- Two types:
  - Single rotation (one plane, like 2D/3D)
  - Double rotation (two orthogonal planes)

$$(w + ix + jy + kz)(-i) = x - iw - jz + ky$$

-90° rotation in (w, x) plane

90° rotation in (y, z) plane

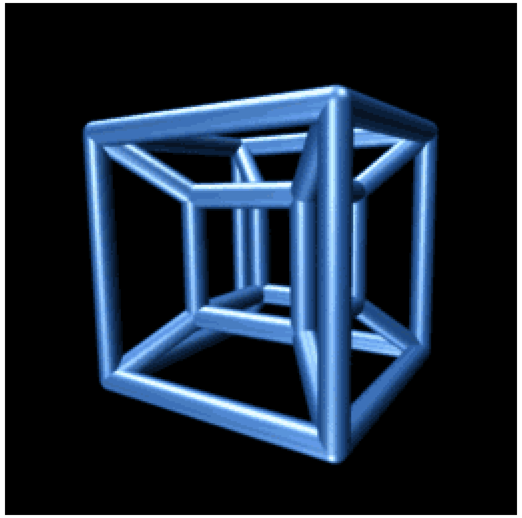
4D Rotation

- Two types:
  - Single rotation (one plane, like 2D/3D)
  - Double rotation (two orthogonal planes)

$$(i)\ (w + ix + jy + kz)(-i) = 0\ +\ 2(-jz + ky)$$

4D Rotation

- Two types:
  - Single rotation (one plane, like 2D/3D)
  - Double rotation (two orthogonal planes)



$$(i)\,(w + ix + jy + kz)(-i) = 0\ +\ 2(-jz + ky)$$

4D Rotation

- Two kinds of quaternion rotations (both angles equal) - **isoclinic**
  - Multiply on left, rotate ccw in both planes
  - Multiply on right, rotate ccw in one, cw in other

4D Rotation

- Matched pair of isoclinic rotations

$$\mathbf{P}' = \mathbf{Q}\mathbf{P}\mathbf{Q}^{-1}$$

Rotate by $\theta/2$ ccw in 3D rotation plane and ccw in the orthogonal plane

Rotate by $\theta/2$ ccw in 3D rotation plane and cw in the orthogonal plane

4D Rotation

- Matched pair of isoclinic rotations

$$\mathbf{P}' = \mathbf{Q}\mathbf{P}\mathbf{Q}^{-1}$$

Rotate by $\theta/2$ ccw in 3D rotation plane ~~and ccw in the orthogonal plane~~

Rotate by $\theta/2$ ccw in 3D rotation plane ~~and cw in the orthogonal plane~~

Quaternion Rotation

- Can easily concatenate rotations – no trigonometric calculations needed

$$\mathbf{q}_1 \cdot (\mathbf{q}_0 \cdot \mathbf{p} \cdot \mathbf{q}_0^{-1}) \cdot \mathbf{q}_1^{-1} = (\mathbf{q}_1 \cdot \mathbf{q}_0) \cdot \mathbf{p} \cdot (\mathbf{q}_1 \cdot \mathbf{q}_0)^{-1}$$

- Note multiplication order: right-to-left

Creating Quaternions

- So for example, if we want to rotate 90° around z-axis:

$$w = \cos(45^\circ) = \sqrt{2}/2$$

$$x = 0 \cdot \sin(45^\circ) = 0$$

$$y = 0 \cdot \sin(45^\circ) = 0$$

$$z = 1 \cdot \sin(45^\circ) = \sqrt{2}/2$$

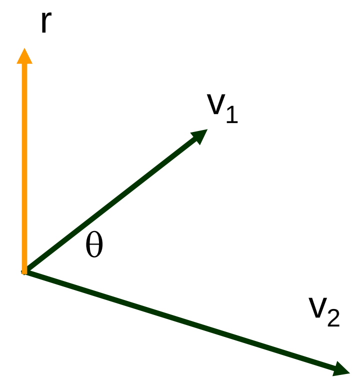
$$\mathbf{q} = (\sqrt{2}/2, 0, 0, \sqrt{2}/2)$$

Creating Quaternions

- Another example
  - We have a vector $\mathbf{v}_1$ and want to rotate to $\mathbf{v}_2$
  - Need rotation axis $\mathbf{r}$, angle θ

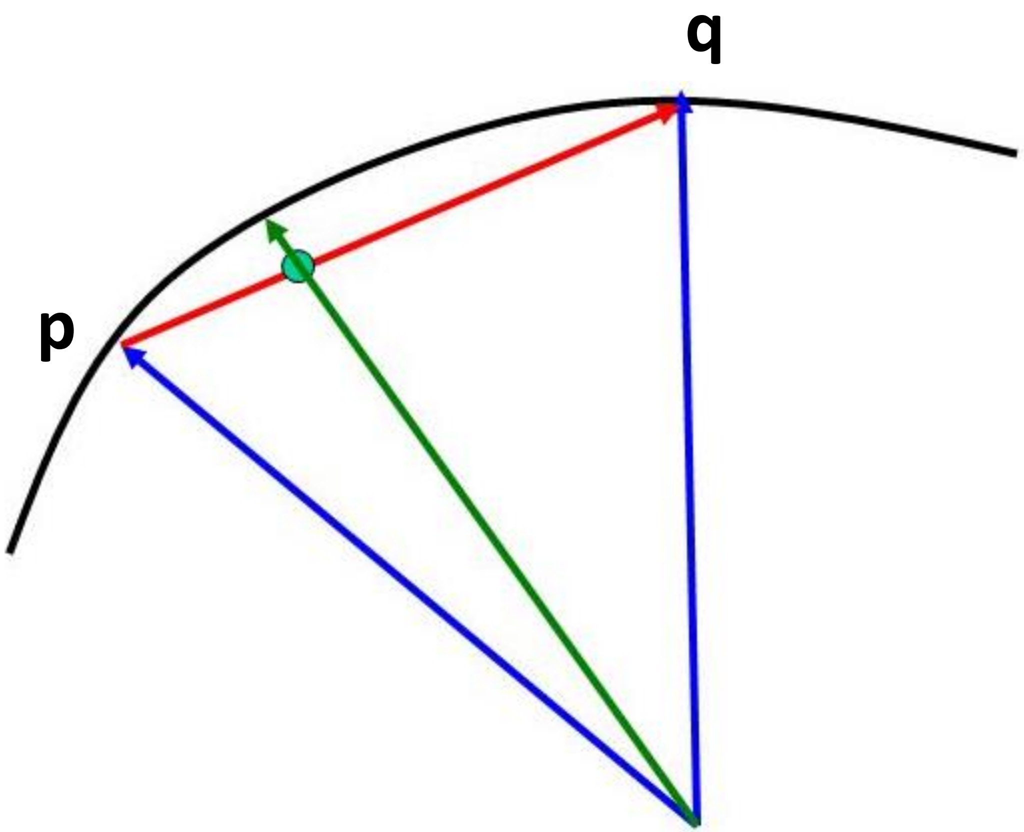
$$\theta = \operatorname{acos}(\hat{\mathbf{v}}_1 \bullet \hat{\mathbf{v}}_2)$$
$$\mathbf{r} = \mathbf{v}_1 \times \mathbf{v}_2$$

  - Plug into previous formula



Interpolating quaternions

- E.g. with linear interpolation: $(1-t)\mathbf{p} + t\mathbf{q}$



Summary

- 2D rotations → unit complex numbers
- 3D rotations → unit quaternions

- Why four values → division algebra
- i, j, k are imaginary axes
- qpq^{-1} is a two step rotation because we can't rotate in a single plane using quaternions in a single step
- Rotation, concatenation and interpolation

glm::gtc::quaternion Namespace Reference

GLM_GTC_quaternion extension: Quaternion types and functions. More...

Typedefs

typedef detail::tquat< double >	**dquat** Quaternion of double-precision floating-point numbers.
typedef detail::tquat< float >	**fquat** Quaternion of single-precision floating-point numbers.
typedef detail::tquat< detail::thalf >	**hquat** Quaternion of half-precision floating-point numbers.
typedef detail::tquat< float >	**quat** Quaternion of floating-point numbers.

Functions

template<typename T > detail::tquat< T >	**conjugate** (detail::tquat< T > const &q) Returns the q conjugate.
template<typename T > detail::tquat< T >	**cross** (detail::tquat< T > const &q1, detail::tquat< T > const &q2) Returns the cross product of q1 and q2.
template<typename T > detail::tquat< T >::value_type	**dot** (detail::tquat< T > const &q1, detail::tquat< T > const &q2) Returns dot product of q1 and q2, i.e., q1[0] * q2[0] + q1[1] * q2[1] + ...
template<typename T > detail::tquat< T >	**inverse** (detail::tquat< T > const &q) Returns the q inverse.