ABSTRACT

This thesis describes various mathematical rotations representations, including 2D rotation matrices, complex numbers, Euler angles, 3D rotation matrices (parametrized by either Euler angles or by an axis and angle), unit quaternions, and axis-angle vectors with exp and log maps. Different conventions for writing and understanding rotations, such as active and passive perspectives, intrinsic and extrinsic rotations, handedness and types of coordinate systems, are explained. The thesis proves that intrinsic rotations are the reverse order of extrinsic rotations, and that composition of quaternion rotations is achieved by quaternion multiplication. Axis-angle 3D rotation matrices are derived using a geometric construction, along with a method for extracting their angle and axis. The Euler-Rodriguez formula is shown to be present in both axis-angle matrices and unit quaternion rotations. The relationship between the exp map and the real exponential function e^x is explained. Furthermore, the thesis describes topics such as gimbal lock, the ambiguity of different representations, and orientation interpolation ($SLERP$ is derived and compared with $NLERP$).

STRESZCZENIE

Praca przedstawia różne matematyczne reprezentacje obrotów, takie jak macierze rotacji 2D, liczby zespolone, kąty Eulera, macierze rotacji 3D (parametryzowane za pomocą kątów Eulera albo osi i kąta), kwaterniony jednostkowe oraz wektory oś-kąt wraz z funkcjami exp i log. Przedstawione są różne konwencje opisu i rozumienia rotacji, takie jak: rotacje aktywne i pasywne, rotacje wokół lokalnych i globalnych osi, skrętność oraz rodzaje układów współrzędnych. Udowodniona jest odwrotna zależność między rotacjami lokalnymi i globalnymi w kątach Eulera oraz fakt, że składanie rotacji kwaternionowych sprowadza się do mnożenia kwaternionów. Macierze rotacji 3D parametryzowane osią i kątem otrzymane są za pomocą konstrukcji geometrycznej, wraz z metodą ekstrakcji osi i kąta obrotu. Pokazane jest, że formuła Eulera-Rodrigueza występuje zarówno w macierzach rotacji 3D parametryzowanych osią i kątem, jak i przy rotacji kwaternionami jednostkowymi. Wyjaśniona jest zależność między funkcją exp a rzeczywistą funkcją e^x. Ponadto praca opisuje takie zagadnienia, jak gimbal lock, wieloznaczność różnych reprezentacji oraz interpolację orientacji ($SLERP$ wraz z porównaniem z $NLERP$).

Contents

Introduction

Almost everyone knows what it means to rotate a physical object with their own hands. It might seem that there's nothing to gain from studying rotations more deeply, especially from studying how to represent them mathematically. However, in practical areas such as 3D animation[14], robotics, or aeronautics[13][11] rotations constitute the most fundamental operations through which virtual characters (both humanoid and animal-like) are animated, robots move, and flying vehicles describe their orientation with respect to Earth. It turns out there are many different ways to represent rotations mathematically. Each representation has its own pros and cons; a skilful user must know when to use which.

Chapter 1 discusses conventions to follow when working with or writing about rotations. In this thesis rotations are understood as active. Vectors are always treated as column vectors. There are two coordinate systems, fixed global one and a changing local one. Both coordinate systems are right-handed. Two right-hand rules apply: for cross product and for positive angle of rotation. Positive angle of rotation is counterclockwise. The **Z** axis is downward, **X** is forward, and **Y** is rightward. Whether intrinsic or extrinsic rotations are performed is never implicit.

Chapter 2 introduces the following topics: Euler's theorem; 2D rotations using 2D rotation matrices and complex numbers; all 24 Euler angles tuples; a proof that intrinsic rotations are the same as extrinsic rotations, but in reversed order; 3D rotation matrices parametrized by Euler angles and by an axis and angle; how to extract axis and angle from 3D rotation matrices; quaternions and how unit quaternions encode and perform rotations; axis-angle vectors with exp and log maps; Euler-Rodriguez formula and its presence in axis-angle matrices and unit quaternion rotations.

Chapter 3 discusses gimbal lock, which occurs when rotations are parameterized by Euler angles, lists ambiguities in different rotation representations, and proves SLERP formula for quaternion interpolation while also comparing it to NLERP.

It's worth noting that for all the presented rotation representations, there

exist conversion formulas to all the other representations. Not all of these formulas are presented in this thesis. Additionally, every representation can be derived in multiple ways: some more intuitive, involving reasoning from analogy and lucky guessing, while others more formal and rigorous.

Chapter 1

Conventions

Before introducing different rotation representations, several conventions need to be established first[1]. The following questions provide an overview of these conventions:

- What is the difference between the terms "rotation" and "orientation"?
- Are vectors represented as rows or columns?
- What types of coordinate systems are used?
- By which symbols are the axes of the coordinate systems denoted?
- Do rotations affect the global coordinate system itself or only the objects defined within it?
- Is the coordinate system right-handed or left-handed?
- Does a positive angle rotation rotate clockwise or counterclockwise when viewed from the tip of the axis towards the origin?
- Which axes are labeled as "forward", "upward", "rightward", etc.?
- When performing a sequence of rotations, are subsequent axes of rotation affected by the previously performed rotations (i.e., are rotations around global fixed axes or local changing axes)?

[1]Some of these conventions are also described in Wikipedia articles: `https://en.wikipedia.org/w/index.php?title=Euler_angles&oldid=1223812775` and `https://en.wikipedia.org/w/index.php?title=Rotation_matrix&oldid=1223594989#Ambiguities` (Date retrieved: 16 June 2024).

These questions will be answered in this chapter. The conventions are somewhat arbitrary, and different texts on the same topic might answer them differently. For the rest of this thesis to be comprehensible, it is of utmost importance to understand these choices.

1.1 Rotation and orientation

The word *rotation* will be understood as an **action**, while the word *orientation* will be understood as a **property/state** of an object. Therefore, objects can be rotated from their original orientation to a new orientation. It is also acceptable to say that orientations are being rotated.

When it comes to mathematical rotation representations, the same entity (e.g., a 3D rotation matrix, a unit quaternion) can represent both an orientation and a rotation, depending on the context. In this thesis, this distinction is made explicit when it is deemed essential; otherwise, "rotation" is used by default.

1.2 Matrix premultiplication and postmultiplication

There are two equivalent ways of transforming vectors using matrices. In the case of **premultiplication**, a column vector $\mathbf{v}$, that is to be transformed, is placed on the right of a transforming matrix $\mathbf{M}$:

$$\begin{aligned} \mathbf{v}' &= \mathbf{M}\mathbf{v} \\ \begin{bmatrix} v'_1 \\ v'_2 \\ \vdots \\ v'_n \end{bmatrix} &= \mathbf{M} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}. \end{aligned} \tag{1.1}$$

In the case of **postmultiplication** the Equations 1.1 are transposed:

$$\begin{aligned} \mathbf{v}'^{\top} &= \mathbf{v}^{\top}\mathbf{M}^{\top} \\ \begin{bmatrix} v'_1 & v'_2 & \cdots & v'_n \end{bmatrix} &= \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \mathbf{M}^{\top} \end{aligned} \tag{1.2}$$

and the column vectors become row vectors, placed on the left of the now-transposed transforming matrix $\mathbf{M}$

Because writing column vectors takes a lot of space (especially in inline equations), they will be written as $\begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}^{\top}$ from now on.

1.3 Coordinate systems

For the purpose of this thesis, two Euclidean[2] (satisfying Euclid's plane geometry axioms) 3D coordinate systems (CS for short) need to be defined: the **global** coordinate system and the **local** coordinate system. The global coordinate system is attached to $\begin{bmatrix}0 & 0 & 0\end{bmatrix}^\top$ and is immovable. The local coordinate system is attached to the global CS's origin and can be rotated around both the global CS's axes and its own axes. Every object has a local coordinate system that defines its orientation with respect to the global CS (see Figure 1.1); the phrases "local CS" and "object's orientation" will be considered synonymous.

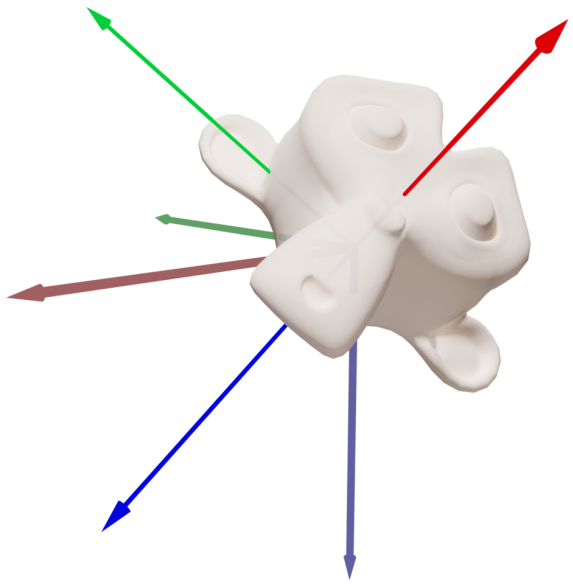


Figure 1.1: Local coordinate system (thin arrows) attached to an object (the head of a monkey) and rotated with respect to the unmoving global coordinate system (thick arrows). The **x** axis (vibrant red) points along the object's forward direction, the **y** axis (vibrant green) points along the object's rightward direction, and the **z** axis (vibrant blue) points along the object's downward direction. The global coordinate system's axes **X**, **Y**, and **Z** are coloured dull red, green, and blue, respectively.

Each CS contains three orthogonal directed axes; because they are directed, they can be represented as three orthogonal unit vectors. Therefore,

[2]While Earth is not strictly Euclidean (if you move forward without ever turning, you'll eventually return to the same place you started), it can locally be modeled as such.

when the word *axis* is mentioned, it should be understood as a vector to which undergraduate linear algebra operations like dot and cross product can be applied. The axes of the global CS will be denoted with $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ vectors, while the axes of the local one will be denoted with $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$.

The identity orientation, both for local and global CSs, will be defined as follows:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^\top & \mathbf{X} &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^\top \\ \mathbf{y} &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^\top & \mathbf{Y} &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^\top \\ \mathbf{z} &= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top & \mathbf{Z} &= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top . \end{aligned}$$

Therefore, in the identity orientation, the local and global CSs overlap. Global CS is immovable, so it is always in the identity orientation.

Whenever an arbitrary vector is defined, it will be with respect to the global CS. Therefore, when a rotation around an arbitrary axis $\mathbf{n}$ is applied, its effect does not depend on the previous rotations applied to the local CS. Similarly, whenever a rotation or orientation is defined, it is defined with respect to the global CS; this prevents different rotations and orientations from being represented by, for example, the same rotation matrix but understood with respect to different CSs.

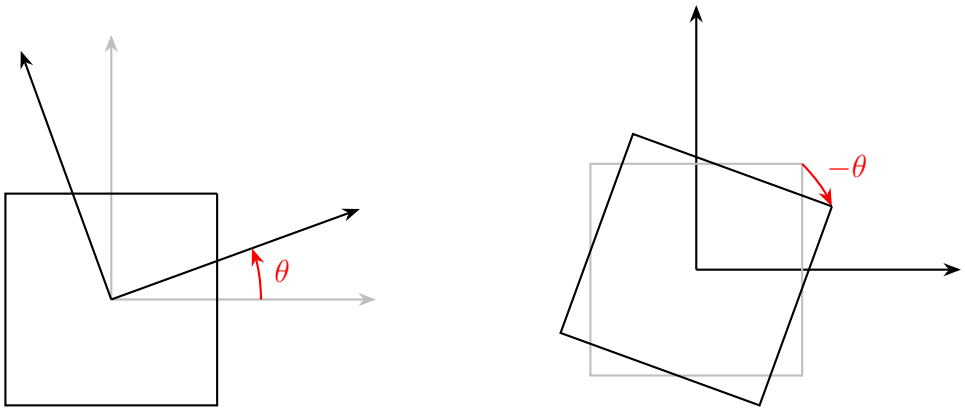
1.4 Active and passive perspectives

There are two perspectives when it comes to performing rotations[3]. These will be defined in terms of Section 1.3. According to the **active** perspective, when performing a rotation, the global CS stays fixed while the local CS changes (see Figure 1.2b). On the other hand, in the **passive** perspective, the global CS changes while the local CS stays fixed (see Figure 1.2a). In Section 1.3 the active perspective has been assumed. This assumption will remain valid for the rest of this thesis.

To convert between these two perspectives, one needs to invert the rotation matrices (or, equivalently, transpose them, as 3D rotation matrices belong to the 3D special orthogonal group $\mathbb{SO}(3)$, which implies that $(\mathbf{R}_1\mathbf{R}_2)^{-1} = (\mathbf{R}_1\mathbf{R}_2)^\top = \mathbf{R}_2^\top\mathbf{R}_1^\top$). If $\mathbf{R}_{active}$ and $\mathbf{R}_{passive}$ are defined as rotation matrices performing the same rotation, but from a different perspective, then the following equation holds:

$$\mathbf{R}_{active} = \mathbf{R}_{passive}^{-1} = \mathbf{R}_{passive}^\top.$$

[3]This dichotomy applies to other linear transformations as well, such as translation, scaling, and shearing. They are, however, ignored here.



(a) Passive rotation (b) Active rotation

Figure 1.2: Every rotation operation can be interpreted either as rotating the global CS (passive rotation) or as rotating the objects (local CSs) contained within the global CS (active rotation). Note that the angle θ changes its sign.

1.5 Handedness, axes directions, and positive angle

The coordinate system is right-handed, with $\mathbf{X}$ (resp. $\mathbf{x}$) axis as the *forward* axis, $\mathbf{Y}$ (resp. $\mathbf{y}$) as the *rightward* axis, and $\mathbf{Z}$ (resp. $\mathbf{z}$) as the *downward* axis in the global (local) CS. The right-hand rule for cross product applies[4]. as does the right-hand rule stating that positive angle of rotation corresponds to counterclockwise rotation[5]. Figure 1.3 illustrates this coordinate system. These conventions also define the spherical coordinates vector[6] (on the unit sphere) to be $\begin{bmatrix}\cos\alpha\sin\beta & \sin\alpha\sin\beta & \cos\beta\end{bmatrix}^\top$ where $0 \leqslant \alpha \leqslant 2\pi$ and $-\frac{\pi}{2} \leqslant$

[4]Let $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{a}\times\mathbf{b}$ be three perpendicular unit-length vectors. The right-hand rule states that in 3D space, $\mathbf{a}$ can be associated with the right hand's index finger pointing forward, $\mathbf{b}$ with the middle finger pointing to the left, and $\mathbf{a}\times\mathbf{b}$ with the thumb pointing upwards. This rule aids in determining the direction of the vector obtained by performing cross product.

[5]Align your right hand's thumb with the axis of rotation and curl the rest of your fingers. Observe that the curl is in the direction of counterclockwise rotation, when viewed from the tip of the axis.

[6]Spherical coordinates provide a way to define a vector in $\mathbb{R}^n$ using angles and radius, in contrast to the default Cartesian coordinates. See Section 2.3.3 for how these coordinates arise from matrix multiplication by special rotation matrices. The choice of these matrices depends on the definition of the coordinate system, specifically the assignment of $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ to the forward, rightward, and downward axes, respectively. Figure 2.2 provides a visual explanation.

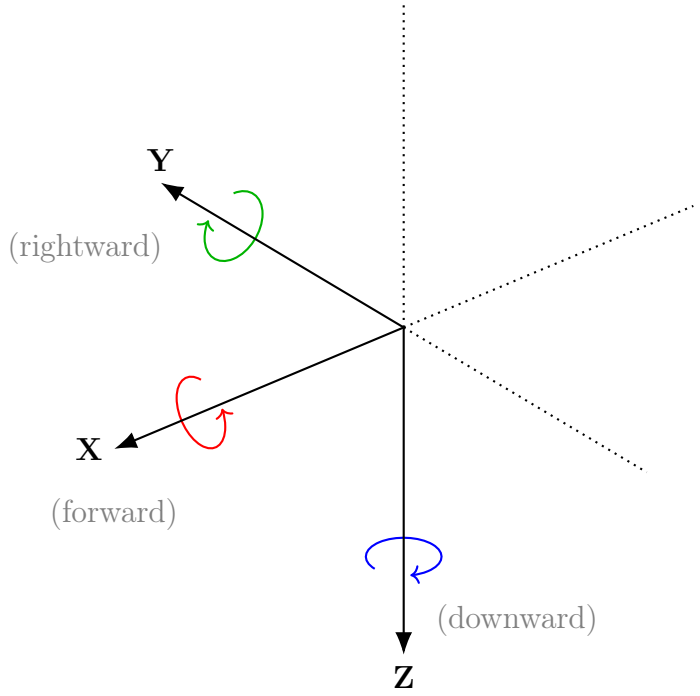


Figure 1.3: Right-handed coordinate system used in this thesis. Positive **X** axis is the *forward* direction, positive **Y** axis is the *rightward* direction, and positive **Z** axis is the *downward* direction. This coordinate system has a nice property: rotation by a positive angle around the **Y** axis causes the **x** axis to go up.

$\beta \leqslant \frac{\pi}{2}$ when starting from the forward axis (or $0 \leqslant \beta \leqslant \pi$ when starting from the downward axis, as discussed in Section 2.3.3).

1.6 Intrinsic and extrinsic rotations

When performing a sequence of rotations three arbitrarily chosen orthogonal axes, it is necessary to specify whether each rotation is performed around the unchanging global axes (**extrinsic** rotations) or around the local axes that change after each rotation (**intrinsic** rotations). In practice, these axes are typically **X**, **Y**, **Z** when rotating around the global CS's axes or **x**, **y**, **z** when rotating around the local CS's axes[7].

[7] `https://en.wikipedia.org/w/index.php?title=Davenport_chained_rotations&oldid=1222677779#Conversion_between_intrinsic_and_extrinsic_rotations` (Date retrieved: 16 June 2024) contains helpful pictures illustrating the step-by-step difference between intrinsic and extrinsic rotations.

Example 1. To rotate an object from its identity orientation (represented by $\mathbf{I}$, the 3 by 3 identity matrix), perform a sequence of rotations: first a 90° rotation around the $\mathbf{X}$, then a 180° rotation around the $\mathbf{Y}$, and finally a 270° rotation around the $\mathbf{Z}$. This sequence is described by the following matrix multiplications[8]:

$$\mathbf{R}_{final} = \mathbf{R}(\mathbf{Z}, \tfrac{3\pi}{2})\mathbf{R}(\mathbf{Y}, \pi)\mathbf{R}(\mathbf{X}, \tfrac{\pi}{2})\mathbf{I}. \tag{1.3}$$

To achieve the same final orientation $\mathbf{R}_{final}$, one can perform similar rotations around the local axes but in reverse order (the proof is given in Section 2.3.2):

$$\begin{aligned} \mathbf{R}_{final} &= \mathbf{R}(\mathbf{x}'', \tfrac{\pi}{2})\mathbf{R}(\mathbf{y}', \pi)\mathbf{R}(\mathbf{z}, \tfrac{3\pi}{2})\mathbf{I} \\ &= \mathbf{R}(\mathbf{x}'', \tfrac{\pi}{2})\mathbf{R}(\mathbf{y}', \pi)\mathbf{R}(\mathbf{Z}, \tfrac{3\pi}{2})\mathbf{I}. \end{aligned} \tag{1.4}$$

The prime symbols in the superscripts of $\mathbf{x}$ and $\mathbf{y}$ signify that the rotation is performed not around the original unchanged local axes but around axes that have been rotated once (by the $\mathbf{z}$ axis for $\mathbf{y}'$) and twice (first by the $\mathbf{z}$ axis and then by the $\mathbf{y}'$ axis for $\mathbf{x}''$). Also note that during the first rotation, the $\mathbf{z}$ and $\mathbf{Z}$ axes are aligned (they have the same direction).

In the rest of this thesis, it will be explicitly stated which perspective is assumed. However, this distinction only matters when describing Euler angles (described in Section 2.2) and sequential rotation matrices (described in Section 2.3.1).

[8]Rotation matrices defined as functions of an axis and an angle $\mathbf{R}(\mathbf{AXIS}, angle)$ have not been defined yet. If confused, read Section 2.3 first.

Chapter 2

Representations

Having introduced conventions, it's time to present different rotation representations. With the exception of Section 2.1 which introduces 2D rotations, 3D rotations are mainly discussed. Higher dimensional rotations are entirely omitted in this thesis.

Section 2.2 introduces *Euler angles* that describe rotations using sequences of rotations around mutually perpendicular axes. Section 2.3.1 expands on Euler angles by defining *sequential rotation matrices*, which perform the same rotations as Euler angles but with a sequence of matrix multiplications. Section 2.3.2 demonstrates that extrinsic rotations are equivalent to intrinsic rotations applied in reverse. In Section 2.3.3, sequential rotation matrices are used to derive *axis-angle matrices*[1], which perform rotations around arbitrary axes by arbitrary angle. Section 2.3.4 introduces the rotation matrix form of the *Euler–Rodrigues formula*[2]. Section 2.3.5 discusses a method of obtaining the axis and angle from rotation matrices.

Section 2.4 introduces quaternions, explaining their basic properties and how unit quaternions encode and perform rotations. It is shown, that Euler-Rodriguez formula is present in the quaternion rotation formula. Section 2.4.3 demonstrates how rotations can be composed using unit quaternions.

Finally, Section 2.5 describes the *axis-angle* representation, along with exp and log maps that, respectively, convert it to and from other rotation representations.

When discussing rotations, it's essential to introduce Euler's theorem, published in 1775 by Leonhard Euler. According to [7], the theorem states:

In whatever way a sphere (rigid body) is turned around its cen-

[1]The terms "sequential rotation matrix" and "axis-angle rotation matrix" are introduced specifically for this thesis and are not standard in the literature.

[2]The significance of this formula is that in its original form, it predates the discovery of quaternions [7]. However, this thesis presents it using modern vector notation.

> *ter. it is always possible to assign it a diameter, whose direction in translated (rotated) position coincides with the position of the beginning.*

("rigid body" and "rotated" in brackets were added by [7] for clarity). In modern terms, this implies that every rotation has an axis around which the rotation occurs. If rotation is viewed as a linear transformation, this axis corresponds to its eigenvector[3]. Euler's theorem also implies that composing two rotations around two arbitrary axes $\mathbf{n}_1$ and $\mathbf{n}_2$ is equivalent to a rotation around some third axis $\mathbf{n}_3$. It's worth keeping this fact in mind when reading about Euler angles and sequential rotation matrices, as these representations do not directly manipulate this third axis.

Another important theme in 3D rotations is their non-commutativity. Imagine two non-parallel axes, $\mathbf{n}_1$ and $\mathbf{n}_2$. First, rotate your hand (or some other physical object) around $\mathbf{n}_1$; secondly, rotate it around $\mathbf{n}_2$. Now, perform these same rotations in reversed order. The orientation that your hand assumes is not the same. This suggests that a proper rotation operator must be non-commutative.

2.1 2D rotations

Before proceeding to 3D rotations, it's worth discussing how to represent 2D rotations. The coordinate system from Figure 1.3 still applies in 2D but with the **Z** axis pointing out of the plane and towards the reader.

If an object of interest is represented as a 2D real vector $\mathbf{v}$, to rotate it counter-clockwise by an angle θ, one can perform the following matrix multiplication:

$$\mathbf{v}' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \mathbf{v}. \tag{2.1}$$

Alternatively, rotations can be represented using complex numbers. Rotating a complex number $x + iy$ by an angle θ can be achieved with the following complex multiplication:

$$(x' + iy') = e^{i\theta}(x + iy) = (\cos\theta + i\sin\theta)(x + iy). \tag{2.2}$$

These two methods are equivalent because the sets of complex numbers and 2D matrices of the form $\left[\begin{smallmatrix} a & -b \\ b & a \end{smallmatrix}\right]$ are isomorphic with respect to complex multiplication and matrix multiplication:

[3] $\mathbf{v}$ is an eigenvector of a linear transformation represented by a matrix $\mathbf{A}$ if $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ for some scalar λ.

$$(a + ib) \cong \begin{bmatrix} a & -b \\ b & a \end{bmatrix},$$

and the restriction to $a = \cos\theta$ and $b = \sin\theta$ is isomorphic, too.

Example 2. To further justify that Equation 2.1 performs a counter-clockwise rotation, note that a 90° rotation rotates the $\mathbf{X}$ axis $\begin{bmatrix} 1 & 0 \end{bmatrix}^\top$ towards the $\mathbf{Y}$ axis $\begin{bmatrix} 0 & 1 \end{bmatrix}^\top$:

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Similarly, it rotates the $\mathbf{Y}$ towards the $-\mathbf{X}$ axis:

$$\begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Everything as expected.

Example 3. To justify that Equation 2.2 performs a counter-clockwise rotation, note that a 90° rotation rotates the $\mathbf{X}$ axis $1 + i0$ towards the $\mathbf{Y}$ axis $0 + i1$:

$$0 + i1 = (0 + i1)(1 + i0) = \left(\cos(\tfrac{\pi}{2}) + i\sin(\tfrac{\pi}{2})\right)(1 + i0).$$

Similarly, it rotates the $\mathbf{Y}$ axis towards the $-\mathbf{X}$ axis:

$$-1 + i0 = (0 + i1)(0 + i1) = \left(\cos(\tfrac{\pi}{2}) + i\sin(\tfrac{\pi}{2})\right)(0 + i1).$$

2.2 Euler angles

Euler angles tuples describe 3D rotations using only three real parameters that represent angles of rotation around either **two** or **three** perpendicular[4] axes[5]. In the general case with 3 perpendicular axes, they can be written as follows:

$$(\alpha_{\mathbf{a}}, \beta_{\mathbf{b}}, \theta_{\mathbf{c}})$$

[4]The generalization of Euler angles to three arbitrary non-parallel non-perpendicular axes is sometimes called "Davenport angles" (`https://en.wikipedia.org/wiki/Davenport_chained_rotations`).

[5]For historical accuracy, [7] suggests referring to the type involving rotation around two axes as "Euler angles" (because Euler used them) and the type involving rotation around three axes as "asymmetric Euler angles", "rotational angles", or "Bryan angles" after George Hartley Bryan. Furthermore, the term "Cardan angles" should be avoided as Gerolamo Cardano had nothing to do with them—it's just a translation error. Additionally, "Tait-Bryan angles" should be discouraged, as Peter Guthrie Tait's contributions were not significant enough. For simplicity, this thesis does not fully adhere to [7]'s suggestions and instead uses the term "Euler angles" for all types of Euler angles.

where **a**, **b**, and **c** represent mutually perpendicular axes, and α, β, and θ represent the angles of rotation. In practice, these axes are not chosen arbitrarily but are either global CS's axes for **extrinsic** rotations, or local CS's axes for **intrinsic** rotations. The tuple $(0_{\mathbf{a}}, 0_{\mathbf{b}}, 0_{\mathbf{c}})$ represents the identity (zero) rotation or orientation.

If the variables **a**, **b**, and **c** are substituted with specific axes such as **X**, **Y**, and **Z**, the following six extrinsic combinations are possible:

$$(\alpha_{\mathbf{X}}, \beta_{\mathbf{Y}}, \theta_{\mathbf{Z}}), (\alpha_{\mathbf{X}}, \beta_{\mathbf{Z}}, \theta_{\mathbf{Y}}), (\alpha_{\mathbf{Y}}, \beta_{\mathbf{X}}, \theta_{\mathbf{Z}}), (\alpha_{\mathbf{Y}}, \beta_{\mathbf{Z}}, \theta_{\mathbf{X}}), (\alpha_{\mathbf{Z}}, \beta_{\mathbf{X}}, \theta_{\mathbf{Y}}), (\alpha_{\mathbf{Z}}, \beta_{\mathbf{Y}}, \theta_{\mathbf{X}}).$$

Example 4. Consider $(\alpha_{\mathbf{X}}, \beta_{\mathbf{Y}}, \theta_{\mathbf{Z}})$: This tuple indicates that an object is first rotated by α degrees around the **X** axis, then by β degrees around the **Y**, and finally by θ degrees around the **Z** axis. The **X**, **Y**, and **Z** axes are global axes so they remain fixed after each subsequent rotation.

On the other hand, if the variables **a**, **b**, and **c** are substituted with local axes **x**, **y**, and **z**, then six intrinsic combinations are possible:

$$(\alpha_{\mathbf{x}}, \beta_{\mathbf{y}}, \theta_{\mathbf{z}}), (\alpha_{\mathbf{x}}, \beta_{\mathbf{z}}, \theta_{\mathbf{y}}), (\alpha_{\mathbf{y}}, \beta_{\mathbf{x}}, \theta_{\mathbf{z}}), (\alpha_{\mathbf{y}}, \beta_{\mathbf{z}}, \theta_{\mathbf{x}}), (\alpha_{\mathbf{z}}, \beta_{\mathbf{x}}, \theta_{\mathbf{y}}), (\alpha_{\mathbf{z}}, \beta_{\mathbf{y}}, \theta_{\mathbf{x}}).$$

Example 5. Consider $(\alpha_{\mathbf{x}}, \beta_{\mathbf{y}}, \theta_{\mathbf{z}})$: This tuple indicates that an object is first rotated by α degrees around the **x** axis, then by β degrees around $\mathbf{y}'$ axis, and finally θ degrees around $\mathbf{z}''$ axis. The meaning of the superscripts was explained in Section 1.6.

All the above Euler angles can also be represented by an alternating rotation using only 2 perpendicular axes

$$(\alpha_{\mathbf{a}}, \beta_{\mathbf{b}}, \theta_{\mathbf{a}}),$$

where 3 rotations are performed, but the first and last rotations are around the same local or global axis[6] By substituting variables for specific axes, additional 12 combinations are obtained:

$$(\alpha_{\mathbf{X}}, \beta_{\mathbf{Y}}, \theta_{\mathbf{X}}), (\alpha_{\mathbf{X}}, \beta_{\mathbf{Z}}, \theta_{\mathbf{X}}), (\alpha_{\mathbf{Y}}, \beta_{\mathbf{X}}, \theta_{\mathbf{Y}}), (\alpha_{\mathbf{Y}}, \beta_{\mathbf{Z}}, \theta_{\mathbf{Y}}), (\alpha_{\mathbf{Z}}, \beta_{\mathbf{X}}, \theta_{\mathbf{Z}}), (\alpha_{\mathbf{Z}}, \beta_{\mathbf{Y}}, \theta_{\mathbf{Z}}),$$
$$(\alpha_{\mathbf{x}}, \beta_{\mathbf{y}}, \theta_{\mathbf{x}}), (\alpha_{\mathbf{x}}, \beta_{\mathbf{z}}, \theta_{\mathbf{x}}), (\alpha_{\mathbf{y}}, \beta_{\mathbf{x}}, \theta_{\mathbf{y}}), (\alpha_{\mathbf{y}}, \beta_{\mathbf{z}}, \theta_{\mathbf{y}}), (\alpha_{\mathbf{z}}, \beta_{\mathbf{x}}, \theta_{\mathbf{z}}), (\alpha_{\mathbf{z}}, \beta_{\mathbf{y}}, \theta_{\mathbf{z}}).$$

[6] `https://en.wikipedia.org/w/index.php?title=Davenport_chained_rotations&oldid=1222677779#Conversion_between_intrinsic_and_extrinsic_rotations` (Date retrieved: 16 June 2024) contains pictures illustrating **zxz** and **ZXZ** rotations.

As already explained in Section 1.6, in order to equate intrinsic and extrinsic rotations, they need to be performed in reverse order:

$$(\alpha_{\mathbf{X}}, \beta_{\mathbf{Y}}, \theta_{\mathbf{Z}}) = (\theta_{\mathbf{z}}, \beta_{\mathbf{y}}, \alpha_{\mathbf{x}})$$
$$(\alpha_{\mathbf{X}}, \beta_{\mathbf{Z}}, \theta_{\mathbf{Y}}) = (\theta_{\mathbf{y}}, \beta_{\mathbf{z}}, \alpha_{\mathbf{x}})$$
$$\dots$$
$$(\alpha_{\mathbf{Z}}, \beta_{\mathbf{Y}}, \theta_{\mathbf{Z}}) = (\theta_{\mathbf{z}}, \beta_{\mathbf{y}}, \alpha_{\mathbf{z}})$$

All in all, there are $(6 + 6) \times 2 = \mathbf{24}$ equivalent[7] ways to represent Euler angles tuples, capable of describing the same set of rotations. Here, 6 correspond to rotations around three axes, another 6 for rotations around two axes, and each multiplied by 2 to account for both intrinsic and extrinsic variants.

Despite their intuitiveness (which is why they are usually the default rotation representation exposed in the graphical user interfaces of various 3D software), Euler angles have several shortcomings. For one thing, to compose them or to extract the axis of rotation mentioned in Euler's theorem (introduced at the beginning of this chapter), one needs to first convert them to 3D sequential rotation matrices (presented in the next section). Even then, the result of multiplying two 3D sequential rotation matrices is not easy to predict[8]. The same can be said of element-wise addition of tuples $(\alpha_{\mathbf{a}}, \beta_{\mathbf{b}}, \theta_{\mathbf{c}}) + (\gamma_{\mathbf{a}}, \omega_{\mathbf{b}}, \phi_{\mathbf{c}}) = ((\alpha{+}\gamma)_{\mathbf{a}}, (\beta{+}\omega)_{\mathbf{b}}, (\theta{+}\phi)_{\mathbf{c}})$, with their result being especially hard to predict when all six angles of both tuples are nonzero. Furthermore, element-wise addition is commutative and produces different results than the non-commutative multiplication of sequential rotation matrices—these operations are not equivalent. Secondly, due to the numerous possible orders, one must always specify which one is being used, increasing the cognitive load. The discussion of the greatest issue with Euler angles, gimbal lock, is postponed until Section 3.1.

Throughout the rest of this thesis, when discussing only the order of axes (regardless of specific angle values), shorthands such as **XYZ**, **XZY**, ...,

[7] To prove this equivalence, it suffices to derive formulas that convert one Euler angles order to another. This can be done by first converting one Euler angles order to 3D rotation matrices or quaternions, and then converting back to the desired order. While former conversion is straightforward (as will be presented in Section 2.3.1 for 3D rotation matrices), the latter is not: it involves algorithms that are not very insightful and require dealing with many singularities and edge cases. Therefore, these algorithms are omitted from this thesis and the reader is instead referred to the work by [2], which presents an efficient method for converting quaternions to Euler angles.

[8] Given two sequential rotation matrices corresponding to different Euler angles (of the same or different order), you compose them using matrix multiplication. Then, what kind of rotation does this resulting matrix perform? If you rotate some object with this matrix, what new orientation does it assume? It's not immediately apparent.

ZYZ will be used for extrinsic rotations, and **xyz**, **xzy**, ..., **zyz** for intrinsic rotations.

2.3 3D rotation matrices

2D rotations (presented in Section 2.1) can be represented by 2 by 2 real orthonormal matrices with determinant 1, belonging to the group $\mathbb{SO}(2)$. 3D rotation matrices are just a generalization to 3 dimensions: they are 3 by 3 real orthogonal matrices with determinant 1, belonging to the special orthogonal group $\mathbb{SO}(3)$. For every matrix $\mathbf{R} \in \mathbb{SO}(3)$, orthogonality implies that $\mathbf{R}\mathbf{R}^\top = \mathbf{I}$, while $\det(\mathbf{R}) = 1$ implies that $\mathbf{R}$ does not represent scaling or reflection transformations, but only a rotation. The fact that $\mathbb{SO}(3)$ is a group means that the composition of two rotations is still a rotation. However, the 2D case differs from the 3D case in that matrix multiplication of 3D rotation matrices is non-commutative.

A very useful and unique feature of 3D rotation matrices is that their columns[9] can be interpreted as the axes $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ of a local CS:

$$\begin{bmatrix} | & | & | \\ \mathbf{x} & \mathbf{y} & \mathbf{z} \\ | & | & | \end{bmatrix}.$$

When inspecting the skeleton in Figure 2.1, this fact allows one to approximately determine the columns of the 3D rotation matrices, which define the orientation of each bone, by visually comparing the directions of the local CS arrows with the directions of the global CS arrows. For example, the orientation of the yellow-coloured spine bone can be approximately described by a matrix whose $\mathbf{y}$ and $\mathbf{z}$ axes have been rotated by 90° degrees counterclockwise around the $\mathbf{X}$ axis (or equivalently $\mathbf{x}$, since in the figure its direction aligns with $\mathbf{X}$'s direction):

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

As mentioned in Section 1.6, in Example 1, a CS in its identity orientation is described by $\mathbf{I}$:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

[9] This is the case when active rotations and matrix premultiplication conventions hold. Otherwise, it might be rows instead of columns.

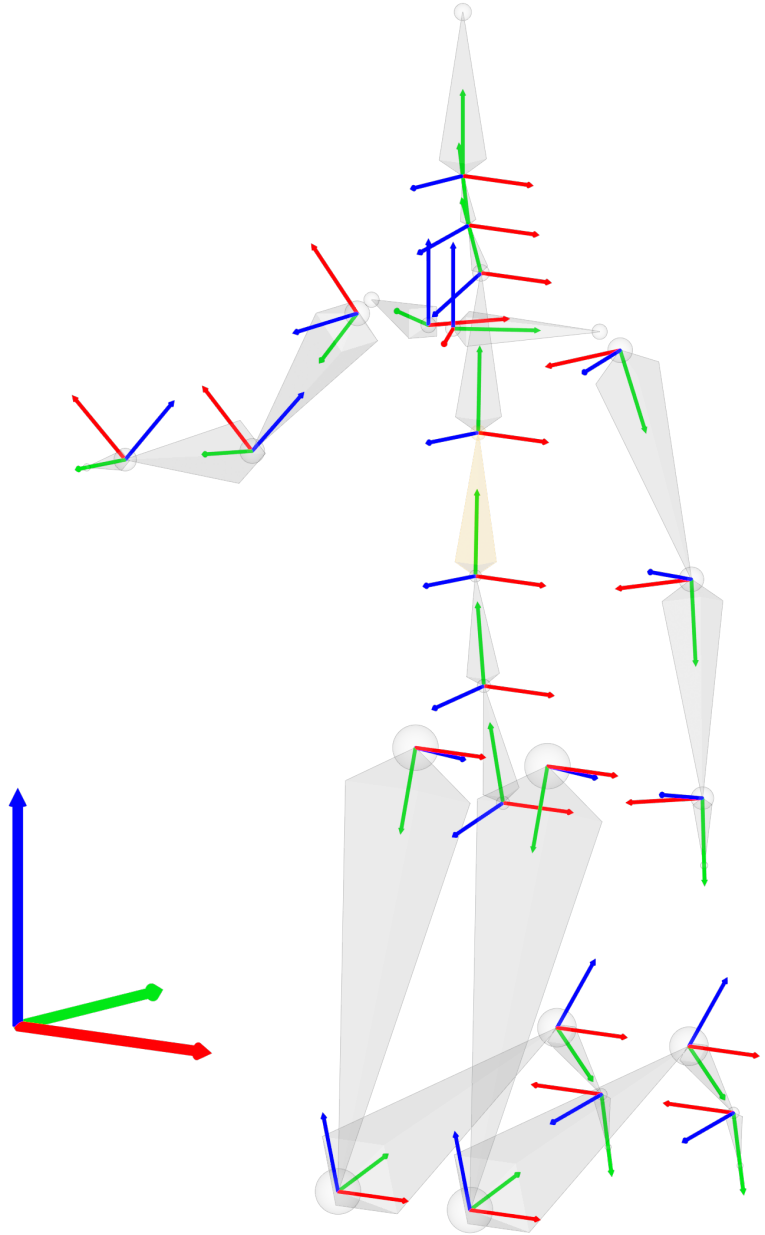


Figure 2.1: A kneeling 3D human skeleton, used to animate meshes of 3D characters. The local CS of every bone is visualized with red (**x**), green (**y**), and blue (**z**) arrows. In this figure, the green arrows point along the length of the bones; this choice, however, is arbitrary. The global CS is visualized on the left with three big arrows; it is different from the global CS used in this thesis—its **Z** axis is upwards, not downwards. The local CSs are translated (the fact that they have different origins than the global CS) only for clarity.

2.3.1 Sequential rotation matrices

Rotation matrices around the three orthogonal axes of the global CS[10] are defined as follows:

$$\mathbf{R}(\mathbf{X},\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{R}(\mathbf{Y},\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}(\mathbf{Z},\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Intuitively, these three matrices are analogous to the 2D rotation matrix. Instead of rotating only the **XY** plane, they rotate (in 3D space) the **YZ** plane, the **ZX**, and the **XY** plane, respectively. This intuition helps explain why $\mathbf{R}(\theta)$ appears in the upper-left corner of $\mathbf{R}(\mathbf{Z},\theta)$.

Example 6. Note that multiplying an arbitrary vector $\begin{bmatrix} a & b & c \end{bmatrix}^\top$ by these rotation matrices further supports this intuition:

$$\begin{bmatrix} a \\ b\cos\theta - c\sin\theta \\ b\sin\theta + c\cos\theta \end{bmatrix} = \mathbf{R}(\mathbf{X},\theta) \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

leaves the x component unchanged,

$$\begin{bmatrix} a\cos\theta + c\sin\theta \\ b \\ -a\sin\theta + c\cos\theta \end{bmatrix} = \mathbf{R}(\mathbf{Y},\theta) \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

leaves the y component unchanged,

$$\begin{bmatrix} a\cos\theta - b\sin\theta \\ a\sin\theta + b\cos\theta \\ c \end{bmatrix} = \mathbf{R}(\mathbf{Z},\theta) \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

leaves the z component unchanged.

[10] To perform intrinsic rotations, one needs to calculate the matrices corresponding to local axes (the ones with superscripts, like $\mathbf{y}'$ or $\mathbf{x}''$). Section 2.3.2 discusses how to construct these matrices.

Example 7. To verify that these three matrices rotate counter-clockwise (as depicted in Figure 1.3), observe that:

$$\begin{aligned}\mathbf{Y} &= \mathbf{R}(\mathbf{X}, -\tfrac{\pi}{2})\mathbf{Z} \\ &= \mathbf{R}^\top(\mathbf{X}, \tfrac{\pi}{2})\mathbf{Z} \\ &= \mathbf{R}(\mathbf{X}, \tfrac{3\pi}{2})\mathbf{Z} \\ &= \mathbf{R}^\top(\mathbf{X}, -\tfrac{3\pi}{2})\mathbf{Z}\end{aligned} \qquad \begin{aligned}\mathbf{Y} &= \mathbf{R}(\mathbf{Z}, \tfrac{\pi}{2})\mathbf{X} \\ &= \mathbf{R}^\top(\mathbf{Z}, -\tfrac{\pi}{2})\mathbf{X} \\ &= \mathbf{R}(\mathbf{Z}, -\tfrac{3\pi}{2})\mathbf{X} \\ &= \mathbf{R}^\top(\mathbf{Z}, \tfrac{3\pi}{2})\mathbf{X}\end{aligned} \qquad \begin{aligned}\mathbf{Z} &= \mathbf{R}(\mathbf{Y}, -\tfrac{\pi}{2})\mathbf{X} \\ &= \mathbf{R}^\top(\mathbf{Y}, \tfrac{\pi}{2})\mathbf{X} \\ &= \mathbf{R}(\mathbf{Y}, \tfrac{3\pi}{2})\mathbf{X} \\ &= \mathbf{R}^\top(\mathbf{Y}, -\tfrac{3\pi}{2})\mathbf{X}\end{aligned}$$

(the $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ vectors were defined in Section 1.3).

Multiplying these single-axis rotation matrices produces a *sequential rotation matrix*. Below is an example of such a matrix in **XYZ** order (or equivalently, **zyx** order):

$$\mathbf{R}(\mathbf{Z}, \alpha)\mathbf{R}(\mathbf{Y}, \beta)\mathbf{R}(\mathbf{X}, \theta) = \begin{bmatrix} c_\alpha c_\beta & c_\alpha s_\beta s_\theta - c_\theta s_\alpha & s_\alpha s_\theta + c_\alpha c_\theta s_\beta \\ c_\beta s_\alpha & c_\alpha c_\theta + s_\alpha s_\beta s_\theta & c_\theta s_\alpha s_\beta - c_\alpha s_\theta \\ -s_\beta & c_\beta s_\theta & c_\beta c_\theta \end{bmatrix} \tag{2.3}$$

where $c_?$ = cos(?) and $s_?$ = sin(?). According to conventions followed in this thesis, extrinsic **XYZ** Euler angles correspond to the well-known **yaw-pitch-roll** sequence of rotations[11].

2.3.2 Intrinsic order is extrinsic order reversed

In Section 1.6, Example 1 mentioned that the intrinsic order is the reverse of the extrinsic order. In matrix notation, using **ZYX** order as an example, this means that:

$$\mathbf{R}(\mathbf{X}, \alpha)\mathbf{R}(\mathbf{Y}, \beta)\mathbf{R}(\mathbf{Z}, \theta) = \mathbf{R}(\mathbf{z}'', \theta)\mathbf{R}(\mathbf{y}', \beta)\mathbf{R}(\mathbf{x}, \alpha) \tag{2.4}$$

Proof. First, to make equations clearer, make the following assignment:

$$\mathbf{R}(\mathbf{X}, \alpha) = \mathbf{R}_X \qquad \mathbf{R}(\mathbf{Y}, \beta) = \mathbf{R}_Y \qquad \mathbf{R}(\mathbf{Z}, \theta) = \mathbf{R}_Z.$$

Observe that the following equations hold:

$$\begin{aligned}\mathbf{R}(\mathbf{x}, \alpha) &= \mathbf{R}_X \\ \mathbf{R}(\mathbf{y}', \beta) &= \mathbf{R}_X\mathbf{R}_Y\mathbf{R}_X^{-1} \\ \mathbf{R}(\mathbf{z}'', \theta) &= \mathbf{R}_X\mathbf{R}_Y\mathbf{R}_Z\mathbf{R}_Y^{-1}\mathbf{R}_X^{-1}\end{aligned} \tag{2.5}$$

[11]The following Wikipedia article has GIFs that explain it: `https://en.wikipedia.org/w/index.php?title=Aircraft_principal_axes&oldid=1213129487` (Date retrieved: 7 June 2024).

The intuition behind these equations[12] is best explained with $\mathbf{R}(\mathbf{y}', \beta)$ as an example. Since the only available rotation matrices, $\mathbf{R}_X$, $\mathbf{R}_Y$, and $\mathbf{R}_Z$, rotate around global axes (as no other rotation matrices have been defined yet), performing a rotation around the local axis $\mathbf{y}'$ requires:

1. Reverting all the rotations applied so far (reverting $\mathbf{R}_X$ by using $\mathbf{R}_X^{-1}$),
2. Performing a global rotation around $\mathbf{Y}$ (using $\mathbf{R}_Y$),
3. Finally, restoring all the reverted rotations (specifically $\mathbf{R}_X$, by applying $\mathbf{R}_X$ again).

Substituting Equations 2.5 into the right-hand side of Equation 2.4, concludes the proof:

$$\begin{aligned}\mathbf{R}(\mathbf{z}'', \theta)\mathbf{R}(\mathbf{y}', \beta)\mathbf{R}(\mathbf{x}, \alpha) &= (\mathbf{R}_X\mathbf{R}_Y\mathbf{R}_Z\mathbf{R}_Y^{-1}\mathbf{R}_X^{-1})(\mathbf{R}_X\mathbf{R}_Y\mathbf{R}_X^{-1})\mathbf{R}_X \\ &= \mathbf{R}_X\mathbf{R}_Y\mathbf{R}_Z(\mathbf{R}_Y^{-1}\mathbf{R}_X^{-1}\mathbf{R}_X\mathbf{R}_Y)(\mathbf{R}_X^{-1}\mathbf{R}_X) \\ &= \mathbf{R}_X\mathbf{R}_Y\mathbf{R}_Z \\ &= \mathbf{R}(\mathbf{X}, \alpha)\mathbf{R}(\mathbf{Y}, \beta)\mathbf{R}(\mathbf{Z}, \theta). \qquad \square\end{aligned}$$

2.3.3 Axis-angle rotation matrices

Instead of sequentially applying single-axis rotations to reach a final orientation, it may be more convenient to define a rotation matrix by specifying an axis and angle. Here are the steps to obtain it (based on [10, Section 6.2]):

1. Let $\mathbf{n} = \begin{bmatrix}\cos\alpha\sin\beta & \sin\alpha\sin\beta & \cos\beta\end{bmatrix}^\top$, where $0 \leqslant \alpha < 2\pi$ and $0 \leqslant \beta \leqslant \pi$, denote the axis of rotation. Note that:

   $$\mathbf{n} = \begin{bmatrix}\cos\alpha\sin\beta & \sin\alpha\sin\beta & \cos\beta\end{bmatrix}^\top = \mathbf{R}(\mathbf{Z}, \alpha)\mathbf{R}(\mathbf{Y}, \beta)\mathbf{Z}$$

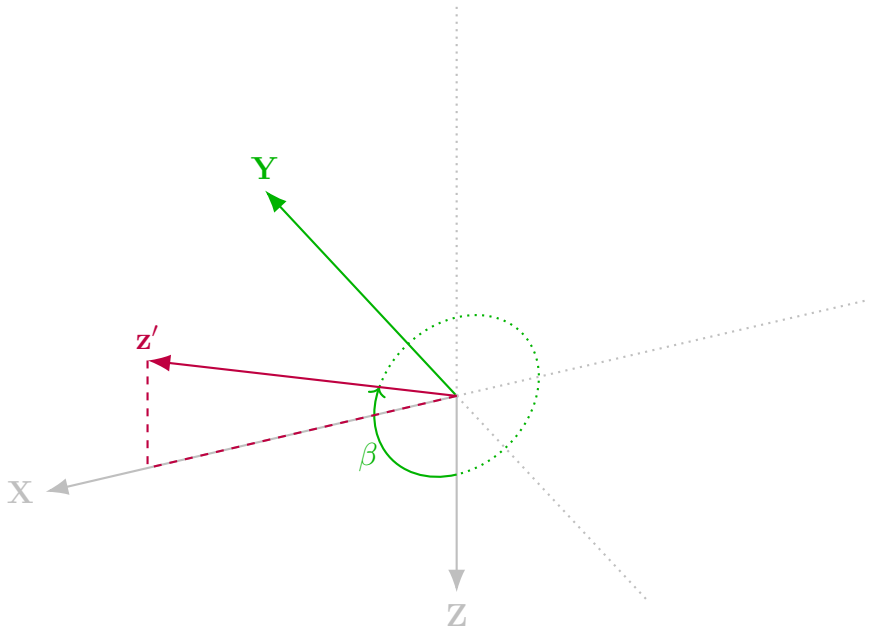
   ($\mathbf{n}$ can alternatively be called $\mathbf{z}''$; see Figure 2.2 for visualization)[13].

2. To construct the rotation matrix that rotates around $\mathbf{n}$, transform $\mathbf{n}$ back to $\mathbf{Z}$ by applying inverse rotations, then rotate around $\mathbf{Z}$ by θ using the $\mathbf{R}(\mathbf{Z}, \theta)$ matrix, and finally rotate $\mathbf{Z}$ back to $\mathbf{n}$:

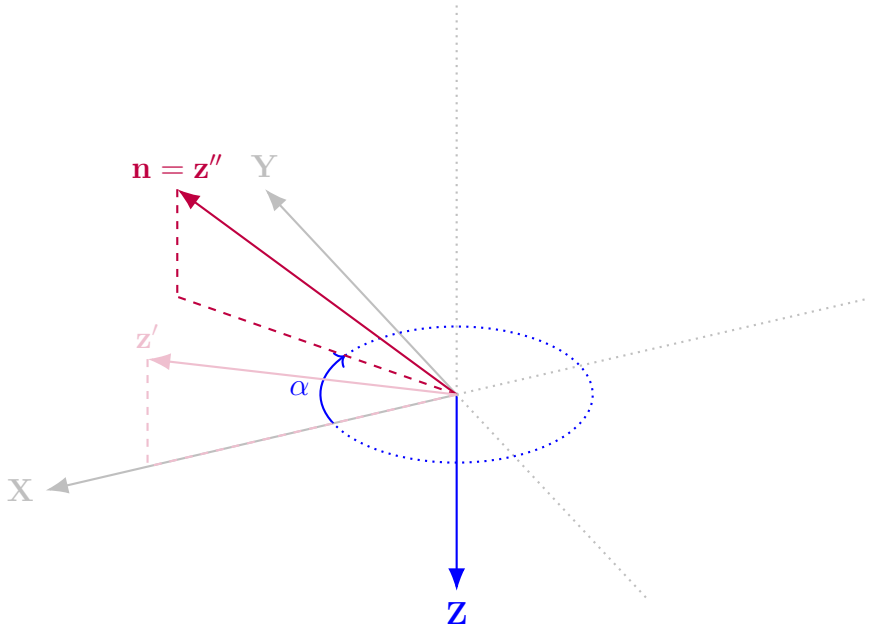
   $$\mathbf{R}(\mathbf{n}, \theta) = \mathbf{R}(\mathbf{Z}, \alpha)\mathbf{R}(\mathbf{Y}, \beta)\mathbf{R}(\mathbf{Z}, \theta)\mathbf{R}^\top(\mathbf{Y}, \beta)\mathbf{R}^\top(\mathbf{Z}, \alpha)$$

[12] More formally, these equations perform a "change of basis". They describe the process of switching to another coordinate system (basis), performing some operation there, and then switching back to the original coordinate system (basis).

[13] Note that if the rightward axis were defined as, for example, $\mathbf{X}$, the downward axis as $-\mathbf{Y}$, then the spherical coordinates vector would be defined as $\mathbf{R}(-\mathbf{Y}, \alpha)\mathbf{R}(\mathbf{X}, \beta)(-\mathbf{Y}) = \begin{bmatrix}\sin\beta\sin\alpha & -\cos\beta & -\sin\beta\cos\alpha\end{bmatrix}^\top$, where $0 \leqslant \alpha < 2\pi$ and $0 \leqslant \beta \leqslant \pi$.



(a) First, a rotation around the **Y** axis by an angle β is performed to obtain $\mathbf{z}'$.



(b) Finally, a rotation around the **Z** axis by an angle α is performed to obtain **n** ($\mathbf{z}''$).

Figure 2.2: Rotation of the axis **z** around the **Y** axis by an angle β, and then rotation of the $\mathbf{z}'$ axis around the **Z** axis by an angle α, finally obtaining the axis **n** ($\mathbf{z}''$). This illustrates the idea behind spherical coordinates.

(it's the same procedure as in Section 2.3.2 but for a different Euler angles order).

3. Perform these 5 matrix multiplications, while keeping in mind that

$$\mathbf{n} = \begin{bmatrix}\cos\alpha\sin\beta & \sin\alpha\sin\beta & \cos\beta\end{bmatrix}^\top = \begin{bmatrix}n_1 & n_2 & n_3\end{bmatrix}^\top.$$

The final result is the *axis-angle rotation matrix*:

$$\mathbf{R}(\mathbf{n},\theta) = \begin{bmatrix} c+(n_1)^2(1-c) & n_1n_2(1-c)-sn_3 & n_1n_3(1-c)+sn_2 \\ n_2n_1(1-c)+sn_3 & c+(n_2)^2(1-c) & n_2n_3(1-c)-sn_1 \\ n_3n_1(1-c)-sn_2 & n_3n_2(1-c)+sn_1 & c+(n_3)^2(1-c) \end{bmatrix} \tag{2.6}$$

where $c = \cos\theta$ and $s = \sin\theta$. It can be verified that $\mathbf{R}(\mathbf{n},\theta)$ is indeed a rotation matrix around $\mathbf{n}$ by confirming that its only real eigenvector is $\mathbf{n}$ with eigenvalue[14] 1, i.e. $\mathbf{R}(\mathbf{n},\theta)\mathbf{n} = \mathbf{n}$ is satisfied. This is intuitively clear, as points lying along $\mathbf{n}$ should not undergo rotation.

2.3.4 Euler-Rodriguez formula

Without any effort, $\mathbf{R}(\mathbf{n},\theta)$ can be decomposed into the following sum:

$$\mathbf{R}(\mathbf{n},\theta) = (\cos\theta)I + (\sin\theta)\left[\mathbf{n}\right]_\times + (1-\cos\theta)(\mathbf{n}\mathbf{n}^\top) \tag{2.7}$$

where $\left[\mathbf{n}\right]_\times$ is the cross product matrix[15] of $\mathbf{n}$:

$$\left[\mathbf{n}\right]_\times = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix},$$

and $\mathbf{n}\mathbf{n}^\top$ is as follows:

$$\mathbf{n}\mathbf{n}^\top = \begin{bmatrix} n_1^2 & n_1n_2 & n_1n_3 \\ n_2n_1 & n_2^2 & n_2n_3 \\ n_3n_1 & n_3n_2 & n_3^2. \end{bmatrix}$$

Equation 2.7 is known as the *Euler-Rodriguez formula* for rotation matrices. Equivalently, it can be reformulated as:

$$\mathbf{R}(\mathbf{n},\theta) = I + (sin\theta)\left[\mathbf{n}\right]_\times + (1-\cos\theta)\left[\mathbf{n}\right]_\times^2.$$

[14]The other two eigenvalues are typically complex for most angles and are not relevant here. Full calculations can be found in the UCSC Physics 116A (Fall 2019) class notes: `https://scipp.ucsc.edu/~haber/ph116A/Rotation2.pdf`.

[15]The name *cross product matrix* stems from the fact that performing cross product $\mathbf{n}\times\mathbf{v}$ is equivalent to multiplying $\left[\mathbf{n}\right]_\times\mathbf{v}$.

2.3.5 Extracting angle and axis

It is possible to deconstruct $\mathbf{R}(\mathbf{n}, \theta)$ (and every other 3D rotation matrix) into its axis $\mathbf{n}$ and its angle θ. To retrieve the angle, it suffices to take the trace:

$$\begin{aligned}\mathrm{Tr}(\mathbf{R}(\mathbf{n}, \theta)) &= (c + (n_1)^2(1-c)) + (c + (n_2)^2(1-c)) + (c + (n_3)^2(1-c)) \\ &= 3c + (n_1^2 + n_2^2 + n_3^2)(1-c) \\ &= 3c + 1 - c \\ &= 1 + 2c = 1 + 2\cos\theta.\end{aligned}$$

After a few algebraic manipulations, the final formula is obtained:

$$\theta = \arccos\left(\frac{\mathrm{Tr}(\mathbf{R}(\mathbf{n}, \theta)) - 1}{2}\right). \tag{2.8}$$

The range of arccos is $[0, \pi]$. To find the axis of rotation, one could, in theory, use the following approach:

$$\begin{aligned}\mathbf{R}(\mathbf{n}, \theta) - \mathbf{R}^\top(\mathbf{n}, \theta) &= \begin{bmatrix} 0 & -2sn_3 & 2sn_2 \\ 2sn_3 & 0 & -2sn_1 \\ -2sn_2 & 2sn_1 & 0 \end{bmatrix} \\ &= 2\sin\theta \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix} \\ &= 2\sin\theta \, [\mathbf{n}]_\times\end{aligned}$$

$$\mathbf{n} = \frac{1}{2\sin\theta} \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix} \tag{2.9}$$

where $R = \mathbf{R}(\mathbf{n}, \theta)$. However, this method won't work for $\theta = \pi$ due to division by zero; the case when $\theta = 0$ is not problematic—it simply represents the identity rotation.

2.4 Quaternions

Because quaternions are not as well known as 3D matrices, their most important algebraic properties are introduced first. Then, unit quaternions are applied to rotations. Historical details regarding the invention of quaternions involving Sir William Hamilton, Olinde Rodriguez, and Euler won't be discussed here; instead, historical resources like [1] and [7] can be consulted. Other ways of representing quaternions, such as 4 by 4 real matrices or 2 by 2 complex matrices, are not discussed.

2.4.1 Basic facts

Quaternions can be represented either as a sum of the scalar part and the imaginary parts or as a tuple consisting of a scalar and a vector:

$$q = q_0 + iq_1 + jq_2 + kq_3 = (q_0, \mathbf{q}) = (q_0, q_1, q_2, q_3),$$

where q_0 is the scalar part and $\mathbf{q} = (q_1, q_2, q_3)$ is the vector part[16]. The imaginary units satisfy:

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

which implies:

$$\begin{aligned} i &= jk = -kj \\ j &= ki = -ik \\ k &= ij = -ji. \end{aligned}$$

Quaternion multiplication is defined as follows (colors for clarity):

$$\begin{aligned} p * q &= (p_0, p_1, p_2, p_3) * (q_0, q_1, q_2, q_3) \\ &= \begin{pmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_1q_0 + p_0q_1 + p_2q_3 - p_3q_2 \\ p_2q_0 + p_0q_2 + p_3q_1 - p_1q_3 \\ p_3q_0 + p_0q_3 + p_1q_2 - p_2q_1 \end{pmatrix}^\top \\ &= (p_0q_0 - \mathbf{p} \cdot \mathbf{q}, p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p} \times \mathbf{q}). \end{aligned}$$

It is non-commutative due to the presence of the cross product. Furthermore, it is associative, so for quaternions q, p, and s, the following equation holds:

$$q * (p * s) = (q * p) * s.$$

The inner product is defined as:

$$\begin{aligned} p \cdot q &= (p_0, p_1, p_2, p_3) \cdot (q_0, q_1, q_2, q_3) \\ &= p_0q_0 + p_1q_1 + p_2q_2 + p_3q_3 \\ &= p_0q_0 + \mathbf{p} \cdot \mathbf{q}. \end{aligned}$$

Multiplying by a scalar $a \in \mathbb{R}$ is simply:

$$aq = a(q_0, \mathbf{q}) = (aq_0, a\mathbf{q}).$$

[16] q_0 is often denoted as w, q_1 as x, q_2 as y, and q_3 as z.

The conjugation rule is:

$$\begin{aligned}\bar{q} &= (q_0, -q_1, -q_2, -q_3) \\ &= (q_0, -\mathbf{q}).\end{aligned}$$

The identity element is $(1, \mathbf{0})$:

$$\begin{aligned}p * (1, \mathbf{0}) &= (p_0, p_1, p_2, p_3) * (1, 0, 0, 0) \\ &= (p_0 1 - \mathbf{p} \cdot \mathbf{0}, p_0 \mathbf{0} + 1\mathbf{p} + \mathbf{p} \times \mathbf{0}) \\ &= (p_0 - 0, \mathbf{0} + \mathbf{p} + \mathbf{0}) \\ &= (p_0, \mathbf{p}) \\ &= p\end{aligned}$$

$$\begin{aligned}(1, \mathbf{0}) * p &= (1, 0, 0, 0) * (p_0, p_1, p_2, p_3) \\ &= (1 p_0 - \mathbf{0} \cdot \mathbf{p}, 1\mathbf{p} + p_0 \mathbf{0} + \mathbf{0} \times \mathbf{p}) \\ &= (p_0 - 0, \mathbf{p} + \mathbf{0} + \mathbf{0}) \\ &= (p_0, \mathbf{p}) \\ &= p.\end{aligned}$$

To obtain the inverse quaternion q^{-1}, first note that:

$$q * \bar{q} = (q \cdot q, \mathbf{0}) = (\|q\|^2, \mathbf{0}).$$

Dividing both sides by $\|q\|^2$ gives:

$$\begin{aligned}\frac{q * \bar{q}}{\|q\|^2} &= (1, \mathbf{0}) \\ q * \frac{\bar{q}}{\|q\|^2} &= (1, \mathbf{0}),\end{aligned}$$

which shows that:

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

For unit length quaternions satisfying

$$\|q\| = \sqrt{q \cdot q} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = \sqrt{q_0^2 + \mathbf{q} \cdot \mathbf{q}} = 1$$

(lying on the unit hypersphere $\mathbb{S}^3$), the inverse simplifies to:

$$q^{-1} = \bar{q}.$$

2.4.2 Performing rotations

The four components of a unit quaternion can also be represented as

$$q(\mathbf{n}, \theta) = (\cos\tfrac{\theta}{2}, \mathbf{n}\sin\tfrac{\theta}{2}), \tag{2.10}$$

where $\mathbf{n}$ is the axis of rotation and θ its angle ($\frac{\theta}{2}$ is not the angle of rotation!). This can be partially proven by noting that:

$$1 = q_0^2 + (q_1^2 + q_2^2 + q_3^2) = \cos^2\alpha + \sin^2\alpha,$$

therefore

$$\begin{aligned} q_0 &= \cos\alpha \\ \sqrt{q_1^2 + q_2^2 + q_3^2} &= \|\mathbf{q}\| = \sin\alpha, \end{aligned}$$

and finally

$$q = (q_0, \mathbf{q}) = \left(q_0, \frac{\mathbf{q}}{\sqrt{q_1^2 + q_2^2 + q_3^2}}\sqrt{q_1^2 + q_2^2 + q_3^2}\right) = \left(\cos\alpha, \frac{\mathbf{q}}{\|\mathbf{q}\|}\sin\alpha\right).$$

There are two things left to prove: that α is just $\frac{\theta}{2}$, and that $\mathbf{q}/\|\mathbf{q}\|$ represents the axis $\mathbf{n}$. Due to space constraints, the rest of the proof is omitted.

To rotate a 3D real vector $\mathbf{v}$ by a quaternion $q = q(\mathbf{n}, \theta)$, treat $\mathbf{v}$ as a quaternion with scalar part equal to 0 (such quaternions are called *pure quaternions*), and multiply it from both sides:

$$\begin{aligned} (0, \mathbf{v}') &= q * (0, \mathbf{v}) * q^{-1} \\ &= q(\mathbf{n}, \theta) * (0, \mathbf{v}) * q^{-1}(\mathbf{n}, \theta) \end{aligned} \tag{2.11}$$

The right-hand side expands as follows:

$$\begin{aligned} (0, \mathbf{v}') &= q * (0, \mathbf{v}) * q^{-1} \\ &= (0, (\cos\theta)\mathbf{v} + (\sin\theta)\mathbf{n} \times \mathbf{v} + (1 - \cos\theta)(\mathbf{n} \cdot \mathbf{v})\mathbf{n}). \end{aligned}$$

The vector part is just the Euler-Rodriguez formula[17] applied to $\mathbf{v}$. By extracting this vector part, it becomes evident that this formula matches Equation 2.7:

$$\begin{aligned} \mathbf{v}' &= (\cos\theta)\mathbf{v} + (\sin\theta)\mathbf{n} \times \mathbf{v} + (1 - \cos\theta)(\mathbf{n} \cdot \mathbf{v})\mathbf{n} \\ &= \left((\cos\theta)\mathbf{I} + (\sin\theta)\,[\mathbf{n}]_\times + (1 - \cos\theta)\mathbf{n}\mathbf{n}^\top\right)\mathbf{v} \\ &= \mathbf{R}(\mathbf{n}, \theta)\mathbf{v} \end{aligned}$$

[17] To derive this form, the identity $\mathbf{n} \times (\mathbf{v} \times \mathbf{n}) = (\mathbf{n} \cdot \mathbf{n})\mathbf{v} - (\mathbf{n} \cdot \mathbf{v})\mathbf{n}$ is used. For full calculation, refer to [4].

It's worth noting that during rotations, the quaternions do not need to be unit length, as the quaternion inverse already normalizes them to unit length:

$$q\,(0,\mathbf{v})\,q^{-1} = \frac{q\,(0,\mathbf{v})\,\bar{q}}{\|q\|^2} = \frac{q}{\|q\|}\,(0,\mathbf{v})\,\frac{\bar{q}}{\|q\|}.$$

It also doesn't matter whether q or $-q$ is used:

$$(-q)\,(0,\mathbf{v})\,(-q)^{-1} = (-1)(-1)q\,(0,\mathbf{v})\,q^{-1} = q\,(0,\mathbf{v})\,q^{-1}.$$

Example 8. Because Equation 2.10 holds, the following unit quaternions perform rotations around the global CS's axes:

- $(\cos\theta, \sin\theta, 0, 0)$ rotates by 2θ around $\mathbf{X}$ axis,
- $(\cos\theta, 0, \sin\theta, 0)$ rotates by 2θ around $\mathbf{Y}$ axis,
- $(\cos\theta, 0, 0, \sin\theta)$ rotates by 2θ around $\mathbf{Z}$ axis.

For the proof of Equation 2.10 and Equation 2.11, refer to [13, Section 5.15] and [6]. Briefly stated, this involves constructing a linear transformation that maps vectors to vectors, preserves vector lengths, does not perform reflection, and modifies only the vector component perpendicular to the axis of rotation, leaving the parallel component unchanged.

2.4.3 Composing rotations

Composing rotations with quaternions is straightforward: simply multiply them. Let p and q be unit quaternions. Then, $r = p{*}q$ represents a composite rotation where rotation q is followed by rotation p.

Proof. As a first step, the following auxillary equation needs to be proved

$$(p * q)^{-1} = \frac{\overline{(p * q)}}{\|p * q\|^2} = \frac{\overline{q}}{\|q\|^2} * \frac{\overline{p}}{\|p\|^2} = q^{-1} * p^{-1}. \tag{2.12}$$

Equate the numerators:

$$\begin{aligned}
\overline{(p*q)} &= \begin{pmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ \hline (p_1q_0 + p_0q_1 + p_2q_3 - p_3q_2) \\ \hline (p_2q_0 + p_0q_2 + p_3q_1 - p_1q_3) \\ \hline (p_3q_0 + p_0q_3 + p_1q_2 - p_2q_1) \end{pmatrix}^\top \\
&= \begin{pmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ -p_1q_0 - p_0q_1 - p_2q_3 + p_3q_2 \\ -p_2q_0 - p_0q_2 - p_3q_1 + p_1q_3 \\ -p_3q_0 - p_0q_3 - p_1q_2 + p_2q_1 \end{pmatrix}^\top \\
&= (p_0q_0 - \mathbf{p}\cdot\mathbf{q}, -p_0\mathbf{q} - q_0\mathbf{p} - \mathbf{p}\times\mathbf{q}) \\
&= (q_0p_0 - \mathbf{q}\cdot\mathbf{p}, -p_0\mathbf{q} - q_0\mathbf{p} + \mathbf{q}\times\mathbf{p}) \\
&= (q_0p_0 - (-\mathbf{q})\cdot(-\mathbf{p}), p_0(-\mathbf{q}) + q_0(-\mathbf{p}) + (-\mathbf{q})\times(-\mathbf{p})) \\
&= \overline{q} * \overline{p},
\end{aligned} \tag{2.13}$$

and the denominators, skipping some steps for brevity:

$$\begin{aligned}
\|p*q\|^2 &= \left\| \begin{pmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_1q_0 + p_0q_1 + p_2q_3 - p_3q_2 \\ p_2q_0 + p_0q_2 + p_3q_1 - p_1q_3 \\ p_3q_0 + p_0q_3 + p_1q_2 - p_2q_1 \end{pmatrix}^\top \right\|^2 \\
&= (p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3)^2 \\
&\quad + (p_1q_0 + p_0q_1 + p_2q_3 - p_3q_2)^2 \\
&\quad + (p_2q_0 + p_0q_2 + p_3q_1 - p_1q_3)^2 \\
&\quad + (p_3q_0 + p_0q_3 + p_1q_2 - p_2q_1)^2 \\
&= \dots \\
&= \left(q_0^2 + q_1^2 + q_2^2 + q_3^2\right)\left(p_0^2 + p_1^2 + p_2^2 + p_3^2\right) \\
&= \|q\|^2 \|p\|^2 .
\end{aligned} \tag{2.14}$$

Equations 2.13 and 2.14 together prove Equation 2.12. Then, it is straightforward to show that r represents a rotation by q followed by p:

$$\begin{aligned}
(0, \mathbf{v}') &= r * (0, \mathbf{v}) * r^{-1} \\
&= (p*q) * (0, \mathbf{v}) * (p*q)^{-1} \\
&= p * (q * (0, \mathbf{v}) * q^{-1}) * p^{-1}.
\end{aligned}$$

□

2.5 Axis-angle with exp and log maps

Probably the simplest way to represent a rotation around some axis $\mathbf{n}$ and an angle θ is using either a tuple

$$\left(\begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix}, \theta \right)$$

or a vector

$$\theta \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix}.$$

For example, a 45° rotation around an axis halfway between the $\mathbf{X}$ and $\mathbf{Z}$ axes is as follows:

$$\frac{\pi}{4} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

While elegant, this representation cannot be composed: a sum or dot product of such angle-axis vectors will not yield sensible results (not to mention that proper composition of rotations should be non-commutative). To address this, two special functions, exp and log, can be defined such that they satisfy the following equations:

$$\exp(\theta\mathbf{n}) = q(\mathbf{n}, 2\theta) = (\cos\theta, \mathbf{n}\sin\theta)$$

$$\log(q(\mathbf{n}, 2\theta)) = \log(\exp(\theta\mathbf{n})) = \theta\mathbf{n}$$

$$\log((q(\mathbf{n}, 2\theta))^t) = t\log(q(\mathbf{n}, 2\theta)) \qquad t \in \mathbb{R}$$

$$(q(\mathbf{n}, 2\theta))^t = (\exp(\theta\mathbf{n}))^t = \exp(t\theta\mathbf{n}) = (\cos t\theta, \mathbf{n}\sin t\theta)$$

The angle θ is doubled to compensate for the half-angle in the quaternion's definition in Equation 2.10.

The term "exp" is not coincidental. It can be expanded into a Taylor series, similar to real numbers, yielding a result akin to Euler's formula for complex numbers:

$$\exp(\theta\mathbf{n}) = (1, \mathbf{0}) + \sum_{k=1}^{\infty} \frac{(0, \theta\mathbf{n})^n}{k!} = (\cos\theta, \mathbf{n}\sin\theta) = q(\mathbf{n}, 2\theta),$$

where taking an integer power of a pure quaternion q is equivalent to q performing quaternion multiplication with itself. This equation should become clear upon inspecting a few quaternion multiplications:

$$\begin{aligned}(0,\theta\mathbf{n}) * (0,\theta\mathbf{n}) &= (-\theta^2, 0)\\ (0,\theta\mathbf{n}) * (0,\theta\mathbf{n}) * (0,\theta\mathbf{n}) &= (0, -\theta^3\mathbf{n})\\ (0,\theta\mathbf{n}) * (0,\theta\mathbf{n}) * (0,\theta\mathbf{n}) * (0,\theta\mathbf{n}) &= (+\theta^4, 0).\end{aligned}$$

This sequence reveals the Taylor expansions of $\sin\theta$ and $\cos\theta$ functions.

On the other hand, exp is not the same thing as e^x for $x \in \mathbb{R}$ because it lacks commutativity:

$$\begin{aligned}\exp(\alpha\mathbf{v})\exp(\beta\mathbf{w}) &= q(\mathbf{v}, 2\alpha) * q(\mathbf{w}, 2\beta)\\ &\neq q(\mathbf{w}, 2\beta) * q(\mathbf{v}, 2\alpha)\\ &= \exp(\beta\mathbf{w})\exp(\alpha\mathbf{v}).\end{aligned}$$

Multiplying two exps is not the same as exp of a sum:

$$\begin{aligned}q(\mathbf{v}, 2\alpha) * q(\mathbf{w}, 2\beta) &= \exp(\alpha\mathbf{v})\exp(\beta\mathbf{w})\\ &\neq \exp(\alpha\mathbf{v} + \beta\mathbf{w})\\ &= q(\tfrac{\alpha\mathbf{v}+\beta\mathbf{w}}{\|\alpha\mathbf{v}+\beta\mathbf{w}\|}, 2\,\|\alpha\mathbf{v} + \beta\mathbf{w}\|)\end{aligned} \tag{2.15}$$

This results in an incorrectly composed rotation. Similar counterarguments can be made for log.

One can define similar log and exp maps for $\mathbf{R}(\mathbf{n}, \theta)$ as is demonstrated in [4]. Note, however, that deriving a formula to extract the axis and angle is necessary to define log. For quaternions, this is straightforward. However, to define it for $\mathbf{R}(\mathbf{n}, \theta)$, one needs to use Equation 2.8 for the angle and Equation 2.9 (or a more robust method) for the axis.

For less rigorous derivations of log and exp maps, consult [10, chapters 9 and 18] or the exercise sheet from CMU 15-462/662 (Fall 2021): `http://15462.courses.cs.cmu.edu/fall2021content/exercises/Solutions06.pdf`. More rigorous and general derivations, with references to Lie algebras and Lie groups, can be found in [8].

Chapter 3

Particular topics

This chapter covers miscellaneous topics, that do not share a single overarching theme. Section 3.1 introduces gimbal lock, a common issue related to Euler angles. It demonstrates that gimbal lock is not an inherent issue of Euler angles tuples but rather of parameterizing rotations with three mutually perpendicular axes. Furthermore, the discontinuity of Euler angles is shown. Section 3.2 discusses the ambiguities present in each representation, their injectivity, and whether they are one-to-one with respect to the intuitive notion of "rotation". Section 3.3 examines the possibilities of interpolating two quaternions using $LERP$, $NLERP$ and $SLERP$. Higher-order interpolation is briefly mentioned.

There is one caveat regarding the sections on gimbal lock (Section 3.1) and ambiguities (Section 3.2): they are not directly based on preexisting scientific literature (hence no citations are present in the text). The reader is advised to keep the following potential issues in mind:

- The section on gimbal lock does not put forward bold claims, so the only risk is that its content does not exhaust the topic (for example, there might be cases of gimbal lock that this section ignores). The examples are all verified empirically using symbolic computation or manual tests in 3D software.

- The section on ambiguities presents easily verifiable observations but does not guarantee that the list of ambiguities for each rotation representation is exhaustive. Additionally, this section lacks rigor as it defines ambiguity with respect to the real world instead of focusing solely on mathematical objects and their properties.

3.1 Gimbal lock

A major issue with Euler angles is the so called "gimbal lock"[1]. It can be defined as follows: For some Euler angles tuple $(\alpha_{\mathbf{a}}, \beta_{\mathbf{b}}, \theta_{\mathbf{c}})$, upon setting β to a special value ω, subsequent changes to either α or θ result in a rotation around the same axis $\mathbf{n}$. In other words, for some angle $\gamma \in \mathbb{R}$, the tuple

$$((\alpha + \gamma)_{\mathbf{a}}, \omega_{\mathbf{b}}, \theta_{\mathbf{c}})$$

represents the same rotation as

$$(\alpha_{\mathbf{a}}, \omega_{\mathbf{b}}, (\theta \pm \gamma)_{\mathbf{c}}).$$

The sign of "$\pm$" depends on the value of ω and on the order **abc** (that can be either intrinsic or extrinsic, despite the fact that **abc** is written in lowercase). Example 9 illustrates how gimbal lock becomes apparent in sequential rotation matrices and quaternions.

Example 9. It's easy to verify that $\mathbf{R}(\mathbf{Z}, \alpha)\mathbf{R}(\mathbf{Y}, \beta)\mathbf{R}(\mathbf{X}, \theta)$ (corresponding to the **XYZ** order) experiences gimbal lock when $\beta = \frac{\pi}{2}$:

$$\begin{aligned}
\mathbf{R}(\mathbf{Z}, \alpha)\mathbf{R}(\mathbf{Y}, \tfrac{\pi}{2})\mathbf{R}(\mathbf{X}, \theta) &= \begin{bmatrix} c_\alpha c_{\frac{\pi}{2}} & c_\alpha s_{\frac{\pi}{2}} s_\theta - c_\theta s_\alpha & s_\alpha s_\theta + c_\alpha c_\theta s_{\frac{\pi}{2}} \\ c_{\frac{\pi}{2}} s_\alpha & c_\alpha c_\theta + s_\alpha s_{\frac{\pi}{2}} s_\theta & c_\theta s_\alpha s_{\frac{\pi}{2}} - c_\alpha s_\theta \\ -s_{\frac{\pi}{2}} & c_{\frac{\pi}{2}} s_\theta & c_{\frac{\pi}{2}} c_\theta \end{bmatrix} \\
&= \begin{bmatrix} 0 & c_\alpha s_\theta - c_\theta s_\alpha & s_\alpha s_\theta + c_\alpha c_\theta \\ 0 & c_\alpha c_\theta + s_\alpha s_\theta & c_\theta s_\alpha - c_\alpha s_\theta \\ -1 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & -s_{\alpha-\theta} & c_{\alpha-\theta} \\ 0 & c_{\alpha-\theta} & s_{\alpha-\theta} \\ -1 & 0 & 0 \end{bmatrix}
\end{aligned} \tag{3.1}$$

where $s_? = \sin(?)$ and $c_? = \cos(?)$. After replacing the subtraction of angles with a single letter $\phi = \alpha - \theta$, it becomes apparent that the angles α and θ are responsible for rotations around the same axis. Furthermore, for any angle $\gamma \in \mathbb{R}$, the tuple

$$((\alpha + \gamma)_{\mathbf{X}}, (\tfrac{\pi}{2})_{\mathbf{Y}}, \theta_{\mathbf{Z}})$$

[1] See `https://en.wikipedia.org/wiki/Gimbal` for an explanation of what gimbals are. Euler angles defined around three perpendicular axes are used to model the behavior of the three-gimbal system. [10, Section 2.3] and `https://en.wikipedia.org/w/index.php?title=Gimbal_lock&oldid=1214961199` (Date retrieved: 16 June 2024) mention how gimbal lock was an issue during the Apollo space missions and how gyroscopes and IMUs (inertial measuring units), used at that time to measure the orientation of a spacecraft, relied on Euler angles.

represents the same rotation as

$$(\alpha_{\mathbf{X}}, (\tfrac{\pi}{2})_{\mathbf{Y}}, (\theta - \gamma)_{\mathbf{Z}}).$$

For $\beta = -\frac{\pi}{2}$, gimbal lock is triggered as well, but the sign changes:

$$\begin{aligned}
\mathbf{R}(\mathbf{Z}, \alpha)\mathbf{R}(\mathbf{Y}, -\tfrac{\pi}{2})\mathbf{R}(\mathbf{X}, \theta) &= \begin{bmatrix} c_\alpha c_{\frac{-\pi}{2}} & c_\alpha s_{\frac{-\pi}{2}} s_\theta - c_\theta s_\alpha & s_\alpha s_\theta + c_\alpha c_\theta s_{\frac{-\pi}{2}} \\ c_{\frac{-\pi}{2}} s_\alpha & c_\alpha c_\theta + s_\alpha s_{\frac{-\pi}{2}} s_\theta & c_\theta s_\alpha s_{\frac{-\pi}{2}} - c_\alpha s_\theta \\ -s_{\frac{-\pi}{2}} & c_{\frac{-\pi}{2}} s_\theta & c_{\frac{-\pi}{2}} c_\theta \end{bmatrix} \\
&= \begin{bmatrix} 0 & -c_\alpha s_\theta - c_\theta s_\alpha & s_\alpha s_\theta - c_\alpha c_\theta \\ 0 & c_\alpha c_\theta - s_\alpha s_\theta & -c_\theta s_\alpha - c_\alpha s_\theta \\ 1 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & -s_{\alpha+\theta} & -c_{\alpha+\theta} \\ 0 & c_{\alpha+\theta} & -s_{\alpha+\theta} \\ 1 & 0 & 0 \end{bmatrix}
\end{aligned}$$

where $s_? = \sin(?)$ and $c_? = \cos(?)$. In this case, for any angle $\gamma \in \mathbb{R}$, the tuple

$$((\alpha + \gamma)_{\mathbf{X}}, (-\tfrac{\pi}{2})_{\mathbf{Y}}, \theta_{\mathbf{Z}})$$

represents the same rotation as

$$(\alpha_{\mathbf{X}}, (-\tfrac{\pi}{2})_{\mathbf{Y}}, (\theta + \gamma)_{\mathbf{Z}}).$$

On the other hand, in the **XZY** order, the signs are "flipped": when $\beta = \frac{\pi}{2}$, for any angle $\gamma \in \mathbb{R}$, the tuple

$$((\alpha + \gamma)_{\mathbf{X}}, (\tfrac{\pi}{2})_{\mathbf{Z}}, \theta_{\mathbf{Y}})$$

represents the same rotation as

$$(\alpha_{\mathbf{X}}, (\tfrac{\pi}{2})_{\mathbf{Z}}, (\theta + \gamma)_{\mathbf{Y}}).$$

For $\beta = -\frac{\pi}{2}$,

$$((\alpha + \gamma)_{\mathbf{X}}, (-\tfrac{\pi}{2})_{\mathbf{Z}}, \theta_{\mathbf{Y}})$$

is equivalent to

$$(\alpha_{\mathbf{X}}, (-\tfrac{\pi}{2})_{\mathbf{Z}}, (\theta - \gamma)_{\mathbf{Y}}).$$

Gimbal lock occurs in sequential quaternions as well (some steps are omitted for brevity):

$$\begin{aligned}
& q(\mathbf{Z}, \alpha) * q(\mathbf{Y}, \tfrac{\pi}{2}) * q(\mathbf{X}, \theta) \\
&= (c(\tfrac{\alpha}{2}), \mathbf{Z}s(\tfrac{\alpha}{2})) * (c_{\frac{\pi}{2}}, \mathbf{Y}s(\tfrac{\pi}{2})) * (c(\tfrac{\theta}{2}), \mathbf{X}s(\tfrac{\theta}{2})) \\
&= (c(\tfrac{\alpha}{2}), 0, 0, s(\tfrac{\alpha}{2})) * (0, 0, 1, 0) * (c(\tfrac{\theta}{2}), s(\tfrac{\theta}{2}), 0, 0) \qquad (3.2) \\
&= ... \\
&= \tfrac{\sqrt{2}}{2}(c(\tfrac{\alpha}{2} - \tfrac{\theta}{2}), -s(\tfrac{\alpha}{2} - \tfrac{\theta}{2}), c(\tfrac{\alpha}{2} - \tfrac{\theta}{2}), s(\tfrac{\alpha}{2} - \tfrac{\theta}{2}))
\end{aligned}$$

where $s(?) = \sin(?)$ and $c(?) = \cos(?)$.

Example 9 demonstrates that gimbal lock is not a problem exclusive to Euler angles as opposed to 3D rotation matrices and unit quaternions; rather, it arises from sequentially rotating by a mutually orthogonal set of axes. It is a matter of parametrization, rather than the mathematical objects themselves.

Example 10. Euler angles defined with two perpendicular axes are vulnerable to gimbal lock as well. For example, in the **ZYZ** order, $\mathbf{R}(\mathbf{Z}, \alpha)\mathbf{R}(\mathbf{Y}, \beta)\mathbf{R}(\mathbf{Z}, \theta)$ experiences gimbal lock for $\beta = 0$:

$$\mathbf{R}(\mathbf{Z}, \alpha)\mathbf{R}(\mathbf{Y}, 0)\mathbf{R}(\mathbf{Z}, \theta) = \mathbf{R}(\mathbf{Z}, \alpha)\mathbf{R}(\mathbf{Z}, \theta) = \mathbf{R}(\mathbf{Z}, \alpha + \theta).$$

Both angles become responsible for the same rotation. $\beta = \pi$ and $\beta = -\pi$ are two other assignments that trigger gimbal lock.

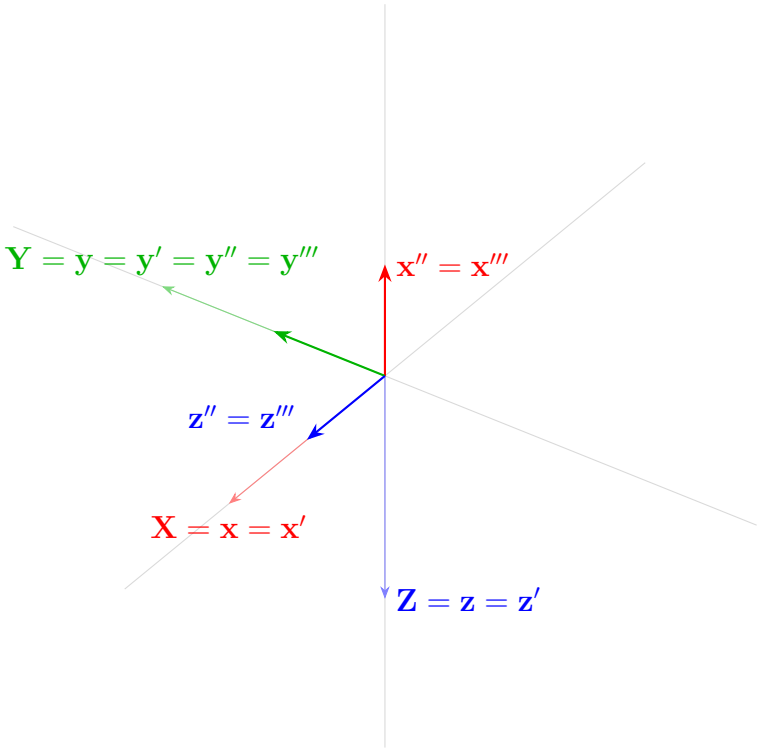


Figure 3.1: The Euler angles tuple $(0^\circ_{\mathbf{z}}, 90^\circ_{\mathbf{y}}, 0^\circ_{\mathbf{x}})$ rotates the initial $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ axes towards the final $\mathbf{x}'''$, $\mathbf{y}'''$, $\mathbf{z}'''$ axes. This tuple is in gimbal lock because $\mathbf{x}'' = -\mathbf{z}$. Axes have different lengths only for clarity—in reality, they are all unit length.

Furthermore, a discontinuity occurs when mapping between some axis-angle parameterized rotations and Euler angles, as demonstrated by Example 11.

Example 11. Suppose that angles α, β, and θ in $(\alpha_{\mathbf{z}}, \beta_{\mathbf{y}}, \theta_{\mathbf{x}})$ Euler angles tuple (**zyx** order) are substituted with values $\alpha = 0°$, $\beta = 90°$, and $\theta = 0°$, to obtain $(0°_{\mathbf{z}}, 90°_{\mathbf{y}}, 0°_{\mathbf{x}})$. Changing α causes a rotation around the $\mathbf{z}$ axis, while changing θ causes a rotation around the $\mathbf{x}''$ axis. However, these axes are parallel—the tuple is in gimbal lock (see Figure 3.1). Because **zyx** tuples perform rotations only around $\mathbf{z}$, $\mathbf{y}'$, and $\mathbf{x}''$ axes, it is impossible to instantaneously (by an infinitesimally small change in the tuple's angles) perform a rotation around the $\mathbf{X}$ axis.

Surprisingly, a rotation by 1° around the $\mathbf{X}$ axis using the axis-angle matrix $\mathbf{R}(\mathbf{X}, 1°)$, rotates the tuple $(0°_{\mathbf{z}}, 90°_{\mathbf{y}}, 0°_{\mathbf{x}})$ towards $(90°_{\mathbf{z}}, 91°_{\mathbf{y}}, 90°_{\mathbf{x}})$ (illustrated in Figure 3.2). This highlights that in gimbal lock, some small axis-angle rotations require significant changes in the Euler angle parameters.

3.2 Ambiguities

There are two kinds of ambiguities worth distinguishing when discussing rotation representations: Sequential rotation matrices, axis-angle rotation matrices, unit quaternions (parameterized by an axis and angle), and axis-angle vectors are all functions—they take some arguments and return some values. If the function is **not injective**, meaning that different arguments map to the same value, this creates one source of ambiguity. The other ambiguity stems from the fact that the same rotation representation can have multiple representatives[2] that all perform the same rotation or represent the same orientation[3]. Because of that, it feels fitting to call it the "**same-thing ambiguity**"[4].

[2]For example, the representatives of the abstract group of 3D rotation matrices are all concrete 3D rotation matrices with each of their 9 values being specific real numbers, not abstract indeterminate variables.

[3]The meaning of "the same rotation" or "the same orientation" could probably be defined more formally, but for the purpose of this section, it doesn't seem worth it.

[4]A good test for same-thing ambiguity is the following scenario: Imagine someone presents you with a physical object and asks, "What is the orientation of this object in the mathematical rotation representation X?". If there is more than one representative of X that corresponds to the presented orientation (e.g., two unit quaternions with different values in their q_0, q_1, q_2, and q_3 components that nonetheless represent the same orientation), then representation X is same-thing ambiguous.

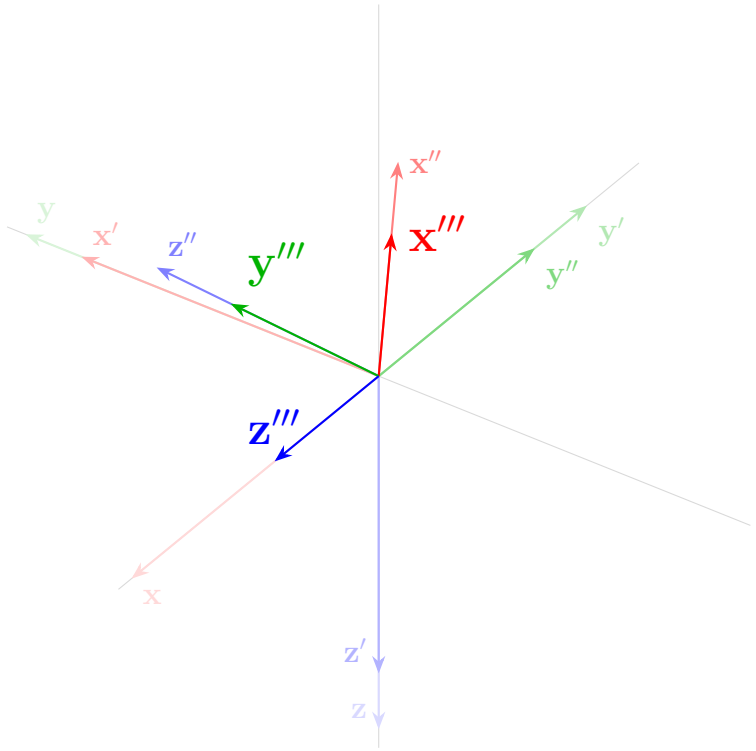


Figure 3.2: Illustration of individual rotations performed by the $(90^\circ_{\mathbf{z}}, 91^\circ_{\mathbf{y}}, 90^\circ_{\mathbf{x}})$ Euler angles tuple, starting from the initial axes $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ to the final axes $\mathbf{x}'''$, $\mathbf{y}'''$, and $\mathbf{z}'''$. The figure actually depicts the $(90^\circ_{\mathbf{z}}, 95^\circ_{\mathbf{y}}, 90^\circ_{\mathbf{x}})$ Euler angles tuple for clarity—91° rotation around $\mathbf{y}'$ would be hard to notice. Moreover, axes have different lengths only for clarity—in reality, they are all unit length.

It is worth pointing out that, contrary to the rotation representations mentioned above, **Euler angles** tuples are not functions—they are just ordered tuples of three real numbers with attached order metadata (**XYZ**, **ZXZ**, etc.). Therefore, same-thing ambiguities are the only type of ambiguities that apply to them. Here is a list of these ambiguities:

- For any $\gamma \in \mathbb{R}$, a gimbal-locked Euler angles tuple $(\alpha_{\mathbf{a}}, \omega_{\mathbf{b}}, \theta_{\mathbf{c}})$ can be expressed either as $((\alpha \pm \gamma)_{\mathbf{a}}, \omega_{\mathbf{b}}, (\theta \mp \gamma)_{\mathbf{c}})$ or as $((\alpha \pm \gamma)_{\mathbf{a}}, \omega_{\mathbf{b}}, (\theta \pm \gamma)_{\mathbf{c}})$, depending on the **a**, **b**, **c**, and ω. Therefore, in gimbal lock there are infinitely many tuples that represent the same rotation.

- Rotations by angles θ and $-(2\pi - \theta)$ are equivalent but represent different Euler angles. For example, $(360^\circ_{\mathbf{X}}, 45^\circ_{\mathbf{Y}}, -180^\circ_{\mathbf{Z}}) = (0^\circ_{\mathbf{X}}, -315^\circ_{\mathbf{Y}}, 180^\circ_{\mathbf{Z}})$. A special case of this is that angles that are multiples of 2π, like 0, -2π,

4π, etc., represent the same rotation.

- Other miscellaneous cases: For example, $(0°_{\mathbf{X}}, 180°_{\mathbf{Y}}, 0°_{\mathbf{Z}})$ is the same rotation as $(180°_{\mathbf{X}}, 0°_{\mathbf{Y}}, 180°_{\mathbf{Z}})$.

When **Axis-angle vectors** are conceived as functions $f(\theta, \mathbf{n}) = \theta\mathbf{n}$, they are not injective because $f(\theta, \mathbf{n}) = f(-\theta, -\mathbf{n}) = \theta\mathbf{n}$. Like Euler angles, they are affected by the same same-thing ambiguity: angles θ and $-(2\pi - \theta)$ are equivalent, but correspond to different axis-angle vectors.

3D rotation matrices do not have same-thing ambiguities: $\mathbf{R}(\mathbf{n}, \theta)$ and $\mathbf{R}(\mathbf{n}, -(2\pi - \theta))$ are the same matrix; $\mathbf{R}(\mathbf{n}, \theta)$ and $\mathbf{R}(-\mathbf{n}, -\theta)$ are the same matrix; there are no other miscellaneous ambiguities. Therefore, one could argue that 3D rotation matrices best model the intuitive notion of "rotation".

However, this does not imply that 3D rotation matrices are injective when parametrized as sequential rotation matrices or as axis-angle matrices: **Sequential rotation matrices**, viewed as functions from three axes and three angles to a rotation matrix, are plagued by the same ambiguities as Euler angles. The difference lies in the fact that in Euler angles these same ambiguities are inherent, whereas in sequential rotation matrices they are a matter of parametrization. **Axis-angle matrices** are not injective for two reasons:

- Similar to Euler angles, $\mathbf{R}(\mathbf{n}, \theta) = \mathbf{R}(\mathbf{n}, -(2\pi - \theta))$.
- Due to the axis-angle parametrization, $\mathbf{R}(\mathbf{n}, \theta) = \mathbf{R}(-\mathbf{n}, -\theta)$.

When it comes to injectivity, **Axis-angle quaternions** $q(\mathbf{n}, \theta)$ are similar to axis-angle rotation matrices. However, they have one additional same-thing ambiguity: quaternions q and $-q$ represent the same rotation (as demonstrated in Section 2.4.2).

Although only mentioned in Section 3.1 in Example 9, sequential unit quaternions are equivalent to sequential rotation matrices in terms of injectivity, and share the same same-thing ambiguities as axis-angle quaternions.

3.3 Interpolation

Interpolation of rotations is an essential feature of every 3D animation software. Instead of specifying an orientation for every frame of the animation, animators can do so only for the so called "key-frames" and rely on the animation software to smoothly interpolate between these key-frames.

In the case of quaternions, such interpolating function takes as its arguments two unit quaternions, a and b, representing the start and end orientations, and a parameter $t \in [0, 1]$, representing the "progress" of the

interpolation. The output of this function is an interpolated orientation. By increasing t from 0 to 1 with constant speed, the interpolated orientation "moves" along the shortest arc between a and b, in $\mathbb{S}^3$. Moreover, this movement is of constant speed (the interpolation progresses at a constant rate). $SLERP$ is the function that satisfies all these requirements, and it is described next.

3.3.1 SLERP

"SLERP" is a term coined by [14] to mean *spherical linear interpolation*. In its most basic form, it is defined as:

$$SLERP(q_0, q_1, t) = \frac{\sin((1-t)\theta)}{\sin(\theta)} q_0 + \frac{\sin(t\theta)}{\sin(\theta)} q_1 \tag{3.3}$$

where q_0 and q_1 are unit quaternions, $t \in [0, 1]$, and $\theta = \arccos(q_0 \cdot q_1)$. Note that the following equations hold:

$$\begin{aligned} SLERP(q_0, q_1, 0) &= q_0 \\ SLERP(q_0, q_1, 1) &= q_1, \end{aligned}$$

and for all $t \in [0, 1]$:

$$SLERP(q_0, q_1, t) = SLERP(q_1, q_0, 1-t).$$

There are a few methods of deriving $SLERP$, but the method by [6], with some clarifications by [10, Chapter 10], is presented here.

Let $q(t) = SLERP(q_0, q_1, t)$, where q_0 and q_1 are unit length quaternions to be interpolated. The quaternion $q(t)$ is located somewhere between q_0 and q_1 depending on the value of t. $q(t)$ can be defined as a linear combination of q_0 and q_1:

$$q(t) = c_0 q_0 + c_1 q_1 \tag{3.4}$$

Let the angle between q_0 and q_1 be denoted as θ. Then, $q(t)$ partitions this angle into θ_0 (the angle between $q(t)$ and q_0) and θ_1 (the angle between $q(t)$ and q_1). Since all the aforementioned quaternions are of unit length, the following set of equations holds:

$$\begin{aligned} \cos\theta &= q_0 \cdot q_1 \\ \cos\theta_0 &= q_0 \cdot q(t) \\ \cos\theta_1 &= q_1 \cdot q(t). \end{aligned} \tag{3.5}$$

Combining Equation 3.4 and Equation 3.5, the following system of equations is obtained:

$$q(t) \cdot q_0 = \cos\theta_0 = c_0 + c_1 \cos\theta$$
$$q(t) \cdot q_1 = \cos\theta_1 = c_0 \cos\theta + c_1$$

It can be elegantly expressed as a matrix equation:

$$\begin{bmatrix} \cos\theta_0 \\ \cos\theta_1 \end{bmatrix} = \begin{bmatrix} 1 & \cos\theta \\ \cos\theta & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}.$$

Solving for c_0 and c_1, the following equations are obtained:

$$c_0 = \frac{\cos\theta_0 - \cos\theta_1 \cos\theta}{1 - \cos^2\theta} = \frac{\cos\theta_0 - \cos\theta_1 \cos\theta}{\sin^2\theta}$$
$$c_1 = \frac{\cos\theta_1 - \cos\theta_0 \cos\theta}{1 - \cos^2\theta} = \frac{\cos\theta_1 - \cos\theta_0 \cos\theta}{\sin^2\theta}$$

Substituting $\theta = \theta_0 + \theta_1$ and applying basic trigonometric identities (specifically, the Pythagorean identity and sum and difference identities) simplifies further to:

$$c_0 = \frac{\cos\theta_0 - \cos\theta_1 \cos(\theta_0 + \theta_1)}{\sin^2\theta} = \frac{\sin\theta_1}{\sin\theta}$$
$$c_1 = \frac{\cos\theta_1 - \cos\theta_0 \cos(\theta_0 + \theta_1)}{\sin^2\theta} = \frac{\sin\theta_0}{\sin\theta}.$$

By substituting $t_0 = \frac{\theta_0}{\theta}$ and $t_1 = \frac{\theta_1}{\theta}$ (where $t_0 + t_1 = \frac{\theta_0}{\theta} + \frac{\theta_1}{\theta} = 1$), and then replacing t_1 with $(1 - t)$ and t_0 with t (which still satisfies $t_0 + t_1 = t + (1 - t) = 1$):

$$c_0 = \frac{\sin(\theta t_1)}{\sin\theta} = \frac{\sin(\theta(1-t))}{\sin\theta}$$
$$c_1 = \frac{\sin(\theta t_0)}{\sin\theta} = \frac{\sin(\theta t)}{\sin\theta}$$

Finally, by substituting c_0 and c_1 into Equation 3.4 , the formula for $SLERP$ (as in Equation 3.3) is derived.

To verify that the speed of $SLERP$ is constant, it suffices to verify that the angle between q_0 and $SLERP(q_0, q_1, t)$ changes at constant rate. In other words, one needs to verify that for all $t \in [0, 1]$:

$$\frac{d}{dt} \arccos(q_0 \cdot SLERP(q_0, q_1, t)) = c \tag{3.6}$$

for some constant $c \in \mathbb{R}$. See Figure 3.3 for an illustration of this idea.

Interestingly, $SLERP(q_0, q_1, t)$ and $SLERP(-q_0, q_1, t)$ are not the same, despite the fact that q_0 and $-q_0$ represent the same rotation. For the orientations to "trace" the shortest arc, it is important that q_0 and q_1 satisfy $q_0 \cdot q_1 \geqslant 0$ [6, Section 4].

[6, Section 4] also introduces an equivalent definition of $SLERP$ that uses exp and log maps. This makes it easy to extend $SLERP$ to 3D rotation matrices [10, Section 10.3].

3.3.2 NLERP

There is a way of interpolating orientations that is not as accurate as $SLERP$, but is simpler and computationally faster. It is called $NLERP$, which stands for *normalized linear interpolation*[5]. $NLERP$ computes *linear interpolation* ($LERP$) on quaternions and normalizes the final result to ensure output quaternions remain unit-length (3D matrices can't be normalized in the same way, therefore $NLERP$ is not applicable to them). $LERP$ for quaternions q and p is defined as[6]

$$\begin{aligned} LERP(q, p, t) = (&q_0(1-t) + p_0 t, \\ &q_1(1-t) + p_1 t, \\ &q_2(1-t) + p_2 t, \\ &q_3(1-t) + p_3 t), \end{aligned}$$

where $t \in [0, 1]$, and $NLERP$ as[7]

$$NLERP(q, p, t) = \frac{LERP(q, p, t)}{\|LERP(q, p, t)\|}.$$

Both satisfy similar equalities as $SLERP$:

$$\begin{aligned} LERP(q, p, 0) &= q \\ LERP(q, p, 1) &= p \\ NLERP(q, p, 0) &= q \\ NLERP(q, p, 1) &= p \end{aligned}$$

[5]Instead of the name $NLERP$, some call it *quaternion linear blending* or *quaternion linear interpolation*.

[6]q_0 and q_1 from section Section 3.3.1 are redefined as q and p to avoid confusion with the quaternion's q_0, q_1, q_2, and q_3 components.

[7]$LERP(q, -q, \frac{1}{2})$ equals $(0, \mathbf{0})$ which in $NLERP(q, -q, \frac{1}{2})$ causes a division by 0. However, this scenario is only theoretical as performing such an interpolation is useless in practice (q and $-q$ represent the same orientation).

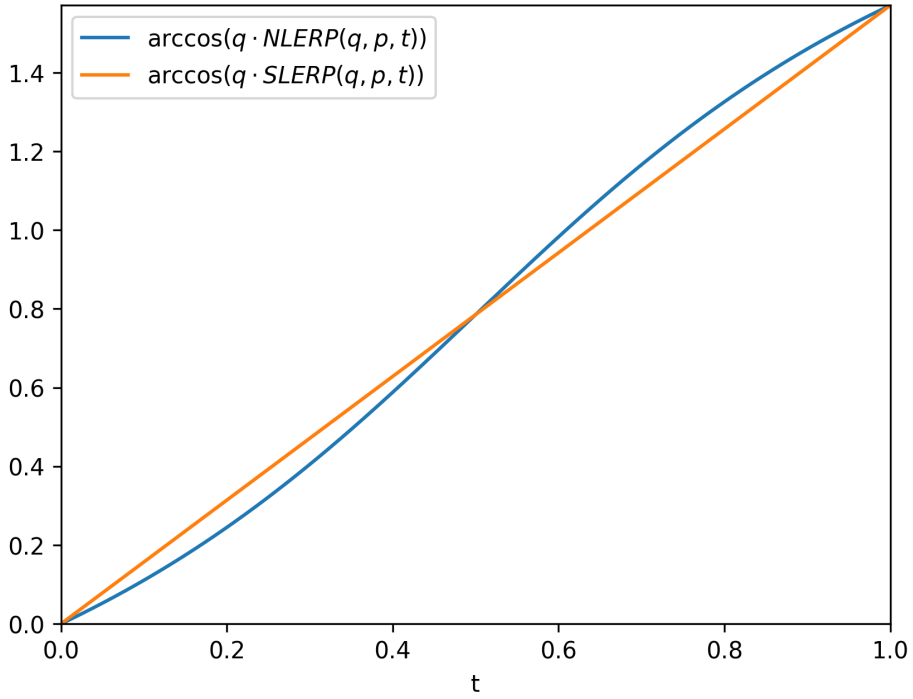


Figure 3.3: The orange curve illustrates the angle between the unit quaternion q and $SLERP(q, p, t)$ for different values of $t \in [0, 1]$. The blue curve shows the same but for $NLERP(q, p, t)$. The plot's Y-axis shows angle values from 0 to π. The unit quaternions q and p are 180° apart (they satisfy $\arccos(q \cdot p) = \pi$), which maximizes the deviation between $SLERP(q, p, t)$ and $NLERP(q, p, t)$. For $t < \frac{1}{2}$, quaternions obtained using $NLERP(q, p, t)$ are closer to q than quaternions obtained using $SLERP(q, p, t)$; for $\frac{1}{2} < t$, the reverse is true.

and for all $t \in [0, 1]$:

$$LERP(q, p, t) = LERP(p, q, 1 - t)$$
$$NLERP(q, p, t) = NLERP(p, q, 1 - t).$$

Surprisingly, when increasing t from 0 to 1 with constant speed, the quaternion $NLERP(q, p, t)$ follows the same curve[8] (in $\mathbb{S}^3$) as $SLERP(q, p, t)$ but with varying speed. However, this variation is not erratic: starting from $t = 0$, $NLERP(q, p, t)$ accelerates until $t = \frac{1}{2}$ when it catches up with $SLERP(q, p, t)$, and then it decelerates until $t = 1$. Figures 3.3 and 3.4 illustrate this speed variation.

[8]This fact is not proven in this thesis. However, it is easy to verify empirically in 3D software.

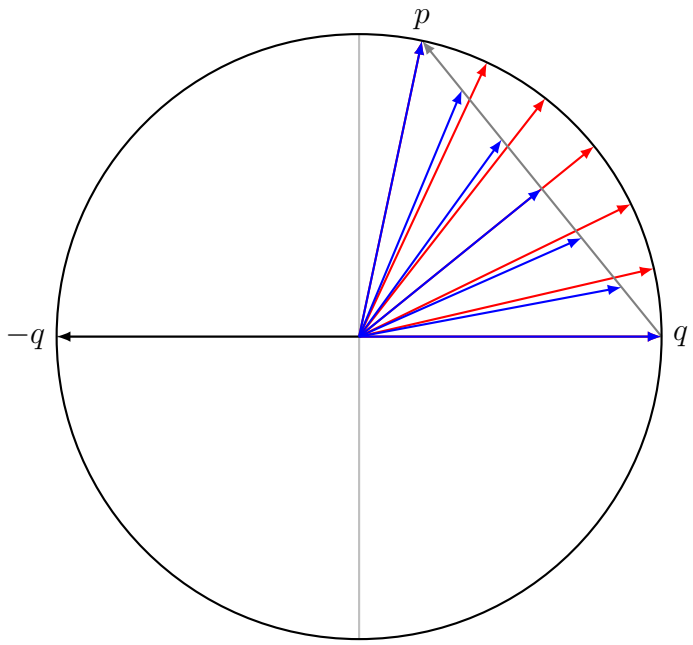


Figure 3.4: Illustrative comparison between $LERP(q, p, t)$ and $SLERP(q, p, t)$ (on $\mathbb{S}^1$ instead of $\mathbb{S}^3$). Red vectors represent $SLERP$ed orientations marking equidistant points on an arc, while blue vectors represent $LERP$ed orientations marking equidistant points on a straight line. Initially, the $LERP$ed vector lags behind the $SLERP$ed vector, catches up in the middle, and leads to the final orientation. Despite the use of 2D vectors, this illustration accurately captures the speed difference between $LERP$ and $SLERP$. If the blue vectors were unit-length, they would represent $NLERP$.

Even when interpolating q and p that are π apart (as in Figure 3.3), the difference between $NLERP$ and $SLERP$ is relatively small. For angles smaller than $\frac{\pi}{2}$ (which is usually the case in practice), the difference is negligible. The viability of $NLERP$ is supported by the fact that the widely known 3D software Blender uses it for quaternion interpolation[9].

If maximum accuracy is required without using $SLERP$, one approach is to transform the parameter t of $NLERP(q, p, t)$ using an S-shaped cubic polynomial $\phi(t) = at^3 + bt^2 + ct + d$, for some $a, b, c, d \in \mathbb{R}$, resulting in $NLERP(q, p, \phi(t))$. This transformation can nullify the speed variations shown in Figure 3.3 and achieve an almost constant speed. This approach is

[9]This is confirmed in the thread from October 2020: `https://devtalk.blender.org/t/quaternion-interpolation/15883/15` . As of June 11 2024, Blender still uses $NLERP$ for quaternion interpolation.

further described in [3].

3.3.3 Higher-order interpolation

Until this point, only methods of interpolating two orientations have been discussed. $SLERP$ could be extended to interpolate between three orientations:

$$SLERP_{curve}(a,b,c,t) = \begin{cases} SLERP(a,b,2t), & \text{if } 0 \leqslant t \leqslant \frac{1}{2} \\ SLERP(b,c,2t-1), & \text{if } \frac{1}{2} < t \leqslant 1 \end{cases} \tag{3.7}$$

where a, b, and c are unit quaternions, and $t \in [0,1]$ (this procedure can be extended to interpolate any number of unit quaternions). However, interpolating more than two orientations using $SLERP_{curve}$ has its shortcomings. As [14] points out:

> *But animations typically have more than two key poses to connect, and here even our spherical elaboration of simple linear interpolation shows flaws. While orientation changes seamlessly, the direction of rotation changes abruptly. In mathematical terms, we want higher order continuity.*

Put another way, $SLERP_{curve}$ is only $\mathcal{C}^0$ continuous: between two orientations a and b (when $0 < t < \frac{1}{2}$), $SLERP_{curve}(a,b,c,t)$ follows a continuous continuous curve that can be differentiated at every point (*orientation changes seamlessly*). However, at the "intersection" of two $SLERP$s at $t = \frac{1}{2}$ (at $SLERP_{curve}(a,b,c,0.5)$), the function is not differentiable. As a result, between $SLERP_{curve}(a,b,c,0.499)$ and $SLERP_{curve}(a,b,c,0.501)$, *the direction of rotation changes abruptly.* To alleviate this, methods with higher order continuity are required (that are $\mathcal{C}^1$, $\mathcal{C}^2$, or even higher). [9] discusses higher-order interpolation in Euclidean space (interpolation of positions) and its generalization to orientations.

Chapter 4

Conclusions

As this thesis has attempted to show, there are many ways to represent rotations, each with its own advantages and disadvantages. It should be clear by now that representing rotations mathematically is no trivial task, and that rotations, while intuitive and common in day-to-day life, conceal depth worth exploring.

Chapter 1 has provided an idea of how different conventions influence the meaning of rotation: where the front is, whether the coordinate system is fixed, whether it is right-handed, whether rotations are around global or local axes, whether rotation matrices are transposed or not, etc. For example, these conventions determine the definition of spherical coordinates and how the elements of a 3D rotation matrix can be interpreted as vectors of a local CS.

Chapter 2 introduced two 2D rotation representations: 2D rotation matrices and complex numbers; and four 3D rotation representations: Euler angles tuples, 3D rotation matrices (parameterized by mutually perpendicular axes and by an axis and angle), unit quaternions, and axis-angle vectors with their exp and log maps.

2D rotations are commutative, but 3D rotations are not. It turns out that different representations are quite interrelated: Euler angles lead to sequential rotation matrices; sequential rotation matrices and axis-angle rotation matrices are simply 3D rotation matrices, just differently parameterized; axis-angle rotation matrices and unit quaternions share a connection to the Euler-Rodriguez formula, and they both share axis-angle parametrization with axis-angle vectors.

Nevertheless, there are some differences: some representations are harder to compose (like Euler angles and axis-angle vectors), some are easy to visualize (like 3D rotation matrices), some can do everything on their own (3D rotation matrices and unit quaternions), while others (Euler angles and

axis-angle vectors) need to be converted to other representations first.

Chapter 3 explored different topics without a single unifying theme. It demonstrated that Euler angles tuples are not the only rotation representation vulnerable to gimbal lock. It highlighted a discontinuity when mapping from axis-angle rotations to sequential rotations. Regarding ambiguity, 3D rotation matrices turned out to be the least ambiguous rotation representation. Proper unit quaternion interpolation is achieved using $SLERP$, although $NLERP$ is a worthy alternative as well. However, higher-order rotation interpolation methods are necessary to smoothly interpolate a sequence of orientations.

It's worth mentioning some topics that were not included in this thesis, but would definitely serve as valuable additions:

- How to perform calculus with rotations? How to compute derivatives of different representations? How to solve differential equations involving rotations?
- How do Lie algebra, Lie groups, and infinitesimal rotations relate to exp and log maps?
- How to construct higher-order rotation interpolation methods?
- Which representations are computationally the fastest and require the least storage?
- Is there a theory that fully describes gimbal lock? When does it occur? How does it relate to the topologies of different rotation representations? What are the ways to avoid it?
- How to generalize Euler angles to non-perpendicular axes, and what are the properties of these generalized Euler angles (*Davenport angles*)?
- Which rotation representation is the most optimal for training deep neural networks that take rotations as their input and/or output?
- How to measure distance between two orientations? In other words, what kind of metrics can be defined[12]?
- Which rotation representation is best for representing rotations by an angle greater than $\pm 180^\circ$ (or even greater than $\pm 360^\circ$)?
- How to perform a swing-twist decomposition of a rotation[5]?

Bibliography

[1] Altmann, Simon L. "Hamilton, Rodrigues, and the Quaternion Scandal". In: *Mathematics Magazine* 62.5 (1989), pp. 291–308. DOI: 10.1080/0025570X.1989.11977459. eprint: https://doi.org/10.1080/0025570X.1989.11977459. URL: https://doi.org/10.1080/0025570X.1989.11977459.

[2] Bernardes, Evandro and Viollet, Stéphane. "Quaternion to Euler angles conversion: A direct, general and computationally efficient method". In: *PLOS ONE* 17.11 (Nov. 2022), pp. 1–13. DOI: 10.1371/journal.pone.0276302. URL: https://doi.org/10.1371/journal.pone.0276302.

[3] Blow, Jonathan. *Hacking Quaternions*. 2002.

[4] Dai, Jian S. "Euler–Rodrigues formula variations, quaternion conjugation and intrinsic connections". In: *Mechanism and Machine Theory* 92 (2015), pp. 144–152. ISSN: 0094-114X. DOI: https://doi.org/10.1016/j.mechmachtheory.2015.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0094114X15000415.

[5] Dobrowolski, Przemyslaw. "Swing-twist decomposition in Clifford algebra". In: *CoRR* abs/1506.05481 (2015). arXiv: 1506.05481. URL: http://arxiv.org/abs/1506.05481.

[6] Eberly, David. "Quaternion Algebra and Calculus". In: (Jan. 2002).

[7] Fraiture, Luc. "A History of the Description of the Three-Dimensional Finite Rotation". In: *The Journal of the Astronautical Sciences* 57 (Mar. 2009). DOI: 10.1007/BF03321502.

[8] Gallier, Jean. "Basics of Classical Lie Groups: The Exponential Map, Lie Groups, and Lie Algebras". In: *Geometric Methods and Applications: For Computer Science and Engineering*. New York, NY: Springer New York, 2001, pp. 367–414. ISBN: 978-1-4613-0137-0. DOI: 10.1007/978-1-4613-0137-0_14. URL: https://doi.org/10.1007/978-1-4613-0137-0_14.

[9] Haarbach, Adrian, Birdal, Tolga, and Ilic, Slobodan. "Survey of Higher Order Rigid Body Motion Interpolation Methods for Keyframe Animation and Continuous-Time Trajectory Estimation". In: *2018 International Conference on 3D Vision (3DV)*. 2018, pp. 381–389. DOI: 10.1109/3DV.2018.00051.

[10] Hanson, Andrew J. *Visualizing Quaternions*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006. ISBN: 9780080474779.

[11] Henderson, D. M. *Euler angles, quaternions, and transformation matrices for space shuttle analysis*. Tech. rep. NASA, June 1977.

[12] Huynh, Du. "Metrics for 3D Rotations: Comparison and Analysis". In: *Journal of Mathematical Imaging and Vision* 35 (Oct. 2009), pp. 155–164. DOI: 10.1007/s10851-009-0161-2.

[13] Kuipers, J. B. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*. Princeton: Princeton University Press, 1999. ISBN: 9780691211701. DOI: doi:10.1515/9780691211701. URL: https://doi.org/10.1515/9780691211701.

[14] Shoemake, Ken. "Animating rotation with quaternion curves". In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '85. New York, NY, USA: Association for Computing Machinery, 1985, pp. 245–254. ISBN: 0897911660. DOI: 10.1145/325334.325242. URL: https://doi.org/10.1145/325334.325242.