Geometric Operators on Manifolds

Practical Implementations for 3D Game Engines

Batool Saleh & Francine Magno

December 2025

Contents

Abstract

Modern game development tools allow people to build complex virtual environments without needing to understand the mathematics behind them. User interfaces are designed to hide this complexity, making it possible to create movement, animation, and interaction through high level controls rather than mathematical reasoning. This background knowledge is not required to use these tools, but it becomes necessary when we want to go beyond what is already provided. Tasks such as designing new movement systems, improving stability, or building custom tools all rely on an understanding of the geometry and mathematics underneath the interface. The goal of this paper is to make that underlying mathematics explicit. Rather than treating movement and orientation as black box operations, we describe them using precise mathematical language. The focus is on connecting common problems in interactive and real time environments to the mathematical tools that are used to solve them.

We begin by modeling movement using standard differential geometry in $\mathbb{R}^3$, where positions and paths can be described using curves, derivatives, and straight line motion. From there, we build the foundation needed to understand more complex transformations later in the paper. The intention is not to replace existing tools, but to explain what they are doing and why. By understanding the mathematics behind these systems, readers are better equipped to extend them, modify them, and build new ones.

[0]Algorithmic and mathematical figures were generated by the author using custom Python and Mathematica code. Conceptual figures were generated with the assistance of AI-based image generation tools, including ChatGPT and Gemini. All source code and figure-generation scripts are available in [9].

1 Movement in $\mathbb{R}^3$

References. This section is based on and informed by the works of [1],[3],[5].

1.1 What Is Movement?

Movement describes how an object's position changes in space over time. When an object moves, every point on the object is displaced by the same amount and in the same direction. The object does not twist, turn, or realign; it is simply translated through space. Because movement only affects position, it can be described entirely in terms of points and vectors in $\mathbb{R}^3$. This makes movement intuitive and mathematically straightforward to model.

1.2 Movement as a Curve in Space

Movement over time can be modeled as a smooth curve

$$\gamma : I \subset \mathbb{R} \to \mathbb{R}^3,$$

where $\gamma(t)$ gives the position of the object at time t. The velocity and acceleration of the object are given by the first and second derivatives of this curve:

$$\dot{\gamma}(t) = \frac{d\gamma}{dt}, \qquad \ddot{\gamma}(t) = \frac{d^2\gamma}{dt^2}.$$

This formulation allows movement to be analyzed using ordinary calculus and differential geometry.

1.3 Geometry of Movement

The space $\mathbb{R}^3$ is flat and Euclidean. Distances, angles, and directions are globally consistent, and there is no curvature to account for. In this setting, geodesics are straight lines. Because the underlying space is flat, the Christoffel symbols vanish, and the geodesic equations reduce to linear motion. This greatly simplifies both analysis and computation.

1.4 Why Ordinary Differential Geometry Is Enough

Since movement occurs in a flat Euclidean space, standard tools from differential geometry are sufficient to describe it. Linear interpolation, velocity integration, and path following can all be performed without introducing ambiguity or error from curvature.

For this reason, the movement functions introduced in the following sections rely only on classical vector calculus and differential geometry in $\mathbb{R}^3$. This provides a clean and efficient foundation before introducing more complex geometric structures later in the paper.

2 ShortestCurve: Why and How Geodesics Appear in Games

References. This section is based on and informed by the works of [3],[5].

In games, "moving straight" should mean moving straight on the surface, not straight in empty 3D space. If a character walks forward on a flat floor, this matches our usual idea of a straight line. But if the same character walks on a planet or other curved surface, moving straight should follow the shape of that surface instead of cutting through it.

When motion is computed using standard Euclidean rules, developers often see issues such as drifting relative to curved geometry, sideways sliding on slopes, or camera paths that feel slightly tilted or unnatural. These problems happen because Euclidean motion ignores surface curvature.

The operator `ShortestCurve()` addresses this by computing geodesics: the shortest paths that lie entirely on a surface. Geodesics provide a surface-aware notion of straight-line motion.

Where this is useful. Geodesic motion appears naturally in many interactive systems:

- Character movement constrained to curved surfaces such as planets.
- Camera paths that move smoothly along spherical or rounded geometry.
- AI navigation on non-flat walkable surfaces.
- Physics-based animation constrained to terrain or meshes.
- Virtual and augmented reality environments.

2.1 Geodesic Setup

Let $X(u, v)$ be a smooth parameterization of a surface. The tangent directions along the surface are given by

$$X_u = \frac{\partial X}{\partial u}, \qquad X_v = \frac{\partial X}{\partial v}.$$

These vectors span the tangent plane at each point on the surface.

The first fundamental form measures distances along the surface:

$$ds^2 = E\,du^2 + 2F\,du\,dv + G\,dv^2,$$

where

$$E = \langle X_u, X_u \rangle, \quad F = \langle X_u, X_v \rangle, \quad G = \langle X_v, X_v \rangle.$$

This defines the surface metric

$$g = \begin{pmatrix} E & F \\ F & G \end{pmatrix}.$$

The Christoffel symbols describe how the coordinate directions change as we move along the surface:

$$\Gamma_{ij}^{k} = \frac{1}{2} g^{k\ell} \big(\partial_i g_{\ell j} + \partial_j g_{\ell i} - \partial_\ell g_{ij} \big).$$

A curve $\gamma(t) = (u(t), v(t))$ is a geodesic if it satisfies

$$\ddot{u} + \Gamma_{11}^{1} \dot{u}^2 + 2\Gamma_{12}^{1} \dot{u}\dot{v} + \Gamma_{22}^{1} \dot{v}^2 = 0,$$

$$\ddot{v} + \Gamma_{11}^{2} \dot{u}^2 + 2\Gamma_{12}^{2} \dot{u}\dot{v} + \Gamma_{22}^{2} \dot{v}^2 = 0.$$

These equations describe motion that stays as straight as possible while remaining on the surface.

2.2 Explicit Case Calculations

2.2.1 Case 1: Plane

Consider the plane

$$X(u, v) = (u, v, 0).$$

Tangents.

$$X_u = (1, 0, 0), \qquad X_v = (0, 1, 0).$$

Metric.

$$E = 1, \quad F = 0, \quad G = 1, \qquad g = I_2.$$

Christoffel symbols. All metric derivatives vanish, so

$$\Gamma_{ij}^{k} = 0.$$

Geodesic equations.

$$\ddot{u} = 0, \qquad \ddot{v} = 0,$$

with solutions

$$u(t) = at + b, \qquad v(t) = ct + d.$$

Interpretation. On a flat surface, geodesics reduce to ordinary straight lines.

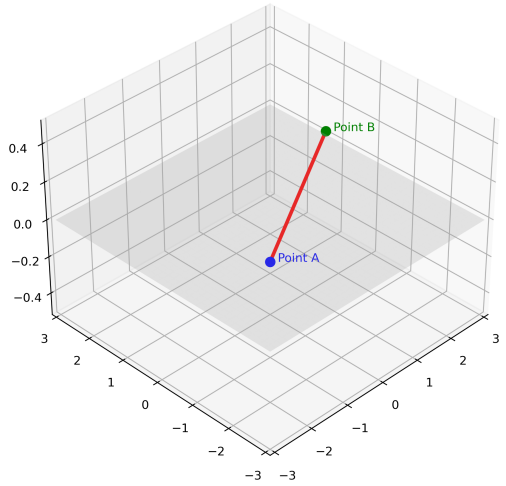

Figure 1: Geodesic on a plane: straight-line motion. Figure generated by the author using custom Python code from [9].

2.2.2 Case 2: Sphere

Now consider the unit sphere parameterized by

$$X(u, v) = (\sin u \cos v,\ \sin u \sin v,\ \cos u).$$

Tangents.

$$X_u = (\cos u \cos v,\ \cos u \sin v,\ -\sin u),$$
$$X_v = (-\sin u \sin v,\ \sin u \cos v,\ 0).$$

Metric.

$$E = 1, \quad F = 0, \quad G = \sin^2 u,$$
$$g = \begin{pmatrix} 1 & 0 \\ 0 & \sin^2 u \end{pmatrix}, \qquad g^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & \csc^2 u \end{pmatrix}.$$

Christoffel symbols. The nonzero symbols are

$$\Gamma^1_{22} = -\sin u \cos u, \qquad \Gamma^2_{12} = \Gamma^2_{21} = \cot u.$$

Geodesic equations.

$$\ddot{u} - \sin u \cos u\, \dot{v}^2 = 0, \qquad \ddot{v} + 2 \cot u\, \dot{u}\dot{v} = 0.$$

Interpretation. On a sphere, geodesics are great circles.

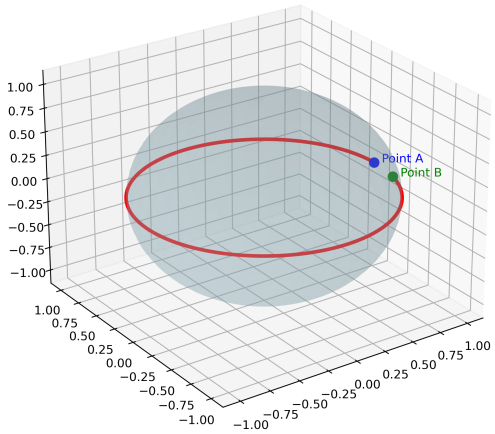

Figure 2: Geodesic on a sphere: great-circle paths. Figure generated by the author using custom Python code from [9].

2.3 Summary

The `ShortestCurve()` operator follows the same steps for all surfaces: compute the metric, derive the Christoffel symbols, and solve the geodesic equations. Flat and spherical surfaces differ only in curvature, not in method. This provides a single, consistent framework for surface-aware motion.

3 CarryNoTwist: Transporting Orientation Without Twisting

References. This section is based on and informed by the works of [1],[3].

3.1 Motivation

When an object moves along a surface, its position is not the only thing that changes. Its orientation (for example, a forward direction or camera axis) must also be updated. A common mistake in game engines is to update orientation using the usual Euclidean derivative in $\mathbb{R}^3$. This often causes objects to slowly rotate or twist even when no turning input is given.

This happens because Euclidean differentiation cannot tell the difference between real rotation and changes caused by the surface bending underneath the object. The goal of `CarryNoTwist()` is to update orientation in a way that avoids this problem. The correct tool for this is *parallel transport*.

3.2 Exterior Derivative vs. Covariant Derivative

Let $\alpha(t)$ be a curve on a surface S and let $W(t)$ be a vector field defined along the curve. In local coordinates, the vector field is written as

$$W(t) = a(t)\,\partial_u + b(t)\,\partial_v.$$

If we differentiate $W(t)$ in the most naive way, we simply differentiate its coordinate components:

$$\frac{dW}{dt} = \dot{a}(t)\,\partial_u + \dot{b}(t)\,\partial_v.$$

This operation treats the basis vectors ∂_u and ∂_v as if they were fixed directions. In other words, it assumes that the coordinate grid does not change as we move along the surface.

This type of differentiation is sometimes called the exterior derivative along the curve. It measures how the coefficients of the vector field change, but it ignores the geometry of the surface itself. On a flat surface, this is fine. However, on a curved surface, the tangent directions ∂_u and ∂_v rotate as we move. Ignoring this effect produces artificial rotation.

The covariant derivative corrects this problem. It differentiates both the coefficients of the vector field and the basis vectors. The change in the basis is encoded by the Christoffel symbols. As a result, the covariant derivative measures how the vector truly rotates within the tangent plane, rather than how it appears to rotate in $\mathbb{R}^3$.

In short:

- The exterior derivative differentiates only the components (a, b) and ignores curvature.
- The covariant derivative accounts for how tangent directions change on the surface.

Parallel transport is defined by setting the covariant derivative to zero, which removes artificial twisting caused by curvature.

3.3 What Parallel Transport Means

Parallel transport is defined by the condition

$$\nabla_{\dot{\alpha}(t)} W(t) = 0.$$

This does not mean that the vector stays fixed in 3D space. Instead, it means that the vector changes only as much as needed to stay tangent to the surface, and no extra rotation is added.

A useful way to think about parallel transport is:

> The vector rotates as little as possible while moving along the surface.

On a flat surface, this means the vector does not rotate at all. On a curved surface, the tangent planes themselves rotate as you move, so a parallel transported vector may appear to rotate in space even though it has not twisted intrinsically.

3.4 Coordinate Form of the Transport Equation

Let the surface be parameterized by coordinates (u, v) and let

$$\alpha(t) = (u(t), v(t)).$$

Write the orientation vector as

$$W(t) = a(t)\,\partial_u + b(t)\,\partial_v.$$

Using the Christoffel symbols Γ^k_{ij}, the condition $\nabla_{\dot{\alpha}} W = 0$ becomes the system

$$\dot{a} = -\Gamma^1_{11} a\dot{u} - \Gamma^1_{12} a\dot{v} - \Gamma^1_{21} b\dot{u} - \Gamma^1_{22} b\dot{v}, \tag{1}$$

$$\dot{b} = -\Gamma^2_{11} a\dot{u} - \Gamma^2_{12} a\dot{v} - \Gamma^2_{21} b\dot{u} - \Gamma^2_{22} b\dot{v}. \tag{2}$$

Existence and uniqueness. This is a first-order linear system with smooth coefficients. By the Picard–Lindelöf theorem, there is a unique solution for any initial orientation $(a(0), b(0))$. Closed-form solutions are generally not needed; in our work, these equations were solved numerically using Python.

3.5 Explicit Case Calculations

3.5.1 Case 1: Plane

Let

$$X(u, v) = (u, v, 0).$$

Metric and Christoffel symbols. The metric is constant, so all Christoffel symbols vanish:

$$\Gamma_{ij}^{k} = 0.$$

Transport equations.

$$\dot{a} = 0, \qquad \dot{b} = 0.$$

Interpretation. The orientation vector stays constant. This matches intuition: moving on a flat surface should not rotate the object.

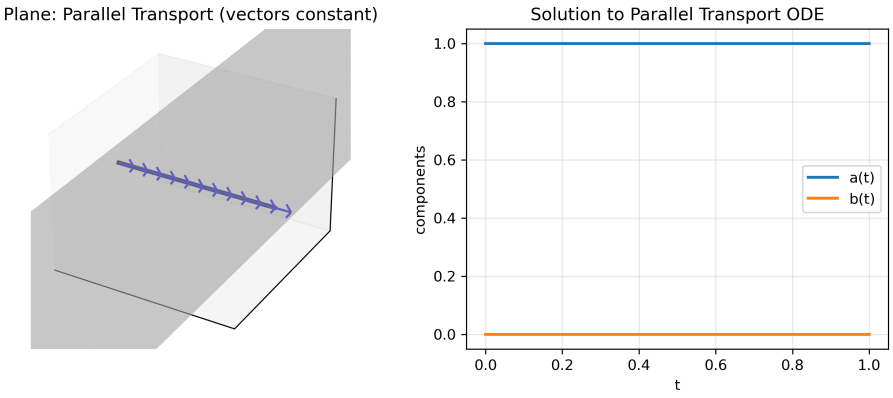

Figure 3: Numerical solution of parallel transport on a plane. The orientation components remain constant. Figure generated by the author using custom Python code from [9].

3.5.2 Case 2: Sphere

Let

$$X(u, v) = (\sin u \cos v,\ \sin u \sin v,\ \cos u).$$

Christoffel symbols. The nonzero symbols are

$$\Gamma_{22}^{1} = -\sin u \cos u, \qquad \Gamma_{12}^{2} = \Gamma_{21}^{2} = \cot u.$$

Transport equations.

$$\dot{a} = \sin u \cos u \, b \, \dot{v}, \tag{3}$$

$$\dot{b} = -\cot u \, a \, \dot{v} - \cot u \, b \, \dot{u}. \tag{4}$$

Interpretation. Even if the object follows a geodesic, its orientation changes due to curvature. This rotation is not an error; it is caused by the geometry of the sphere.

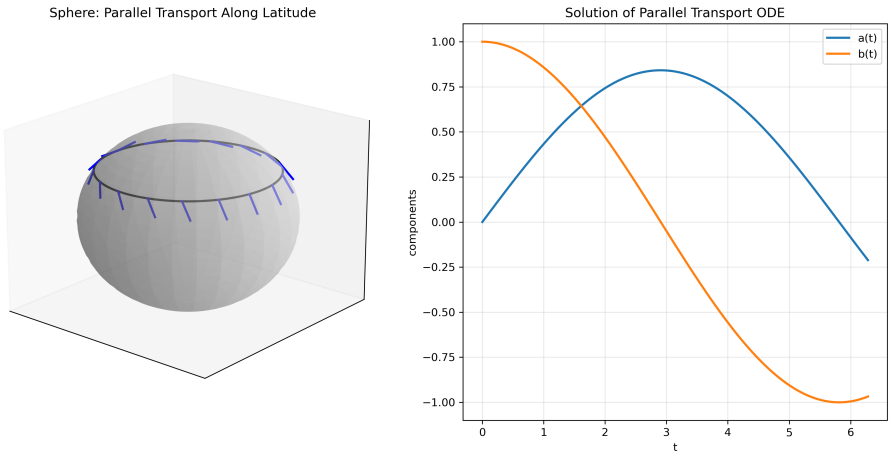

Figure 4: Numerical solution of parallel transport on a sphere. Orientation changes due to curvature even with no intentional turning. Figure generated by the author using custom Python code from [9].

3.5.3 Case 3: Heightmap Terrain

Let

$$h(u,v) = 0.1(u^2+v^2), \qquad X(u,v) = (u,v,h(u,v)).$$

Christoffel symbols. The nonzero symbols simplify to

$$\Gamma^1_{11} = \Gamma^1_{22} = \frac{u}{u^2+v^2+25}, \qquad \Gamma^2_{11} = \Gamma^2_{22} = \frac{v}{u^2+v^2+25}.$$

Transport equations.

$$\dot{a} = -\frac{u}{u^2+v^2+25}(a\dot{u}+b\dot{v}), \tag{5}$$

$$\dot{b} = -\frac{v}{u^2+v^2+25}(a\dot{u}+b\dot{v}). \tag{6}$$

Interpretation. Orientation changes smoothly as the object moves over hills. This explains why objects subtly rotate on uneven ground even when moving straight.

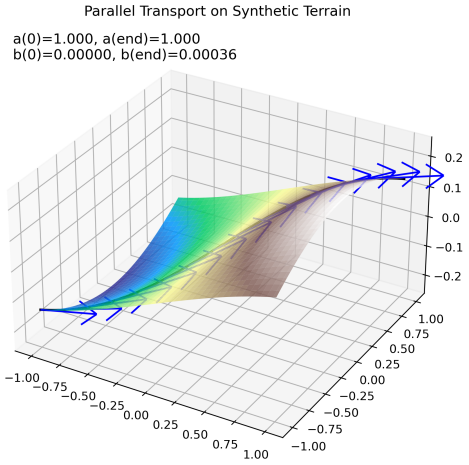

Figure 5: Numerical solution of parallel transport on heightmap terrain. Figure generated by the author using custom Python code from [9].

3.6 Summary

The `CarryNoTwist()` operator updates orientation using the surface's geometry instead of Euclidean differentiation. Parallel transport guarantees that orientation changes are caused only by curvature, not by artificial twisting. Together with `ShortestCurve()`, this provides a complete and geometrically correct model for motion on surfaces.

4 Rotation in $\mathbb{R}^3$

References. This section is based on and informed by the works of [2],[4],[6].

4.1 What Is a Rotation?

A rotation is a rigid motion that changes an object's orientation in space without stretching, bending, or deforming it. Intuitively, it describes how an object turns or spins around a fixed axis. When an object rotates, every point on the object follows a circular path around that axis, while the overall shape and size of the object are preserved.

This makes rotation fundamentally different from translation. A translation moves every point of an object by the same amount and in the same direction, effectively sliding it through space without changing how it is aligned. Rotations, on the other hand, depend on both an axis and an angle, and the final result can vary depending on the order in which rotations are applied.

4.2 Rotation versus Orientation

It is important to distinguish between rotation and orientation. Rotation refers to an action: the transformation that turns an object from one state to another. Orientation, by contrast, is a state: it describes how an object is currently aligned relative to a reference frame. Multiple different rotations can result in the same final orientation, which is one of the reasons rotations are more subtle to work with than translations.

4.3 Mathematical Definition

Mathematically, a rotation in three-dimensional space can be defined as a linear transformation

$$R : \mathbb{R}^3 \to \mathbb{R}^3$$

such that

$$R^T R = I \quad \text{and} \quad \det(R) = 1.$$

The set of all such transformations forms the Lie group SO(3), the special orthogonal group in three dimensions.

The first condition, $R^T R = I$, specifies that R is an orthogonal matrix. This property is crucial as it guarantees that the transformation preserves the standard Euclidean inner product (dot product), which means it conserves both the lengths of vectors and the angles between them.

The second condition, $\det(R) = 1$, is necessary to distinguish a pure rotation from a rotation combined with a reflection. For any orthogonal matrix, the determinant must be ± 1. By enforcing $\det(R) = 1$, we ensure that the transformation preserves the orientation of the coordinate system.

4.4 Euler Angles

One of the most common ways to describe orientation in $\mathbb{R}^3$ is through Euler angles. In this representation, a rotation is described as a sequence of three rotations about coordinate axes:

- **Yaw**: rotation about the y-axis,
- **Pitch**: rotation about the x-axis,
- **Roll**: rotation about the z-axis.

In many 3D graphics engines, the ground plane is defined as the x–z plane with the y-axis pointing upward, making this convention intuitive for cameras and characters. However, different fields and engines use different axis conventions. For example, in aerospace applications, yaw may instead be defined as a rotation about the z-axis. These differences in convention can lead to confusion and errors if not handled carefully.

4.5 What Makes Rotations Different from Movement?

Unlike translations, rotations do not live in a flat vector space. Instead, three-dimensional rotations live on the curved manifold SO(3). Because of this curvature, rotations cannot be treated like ordinary vectors: we cannot freely add or interpolate them using standard linear operations without introducing errors. So To work with rotations properly, we must use tools from the differential geometry of Lie groups.

5 Rotation Matrices and Gimbal Lock

References. This section is based on and informed by the works of [2],[4],[6].

Rotation matrices are one of the most common ways to represent orientation in computer graphics and game engines. They are mathematically well-defined, easy to apply to vectors, and fit naturally into existing transformation pipelines. As a result, they are widely used for object transforms, camera control, and animation.

However, when rotations are constructed using successive rotations about fixed coordinate axes, rotation matrices inherit a fundamental geometric limitation known as gimbal lock. This issue does not come from numerical error or poor implementation. It comes from how rotations are parameterized.

Understanding why gimbal lock occurs is important for explaining why angle-based rotation systems behave poorly in practice and why alternative representations are needed.

5.1 Rotation Matrices

A rotation matrix $R \in \mathbb{R}^{3\times 3}$ satisfies

$$R^T R = I, \qquad \det(R) = 1.$$

Applying a rotation to a vector $\mathbf{x} \in \mathbb{R}^3$ gives

$$\mathbf{x}' = R\mathbf{x}.$$

Rotation matrices preserve lengths and angles. Geometrically, the columns of R form an orthonormal basis representing the rotated coordinate frame. This makes rotation matrices easy to reason about and easy to implement.

Why rotation matrices are widely used. Rotation matrices remain popular in practice for several concrete reasons:

- **Clear geometric meaning.** Each column of the matrix represents a rotated coordinate axis, making it easy to understand how the rotation affects objects and cameras.
- **Direct action on vectors.** Rotating a vector only requires a matrix–vector multiplication, which is computationally cheap and simple to implement.
- **Strong mathematical guarantees.** Orthogonality ensures that lengths and angles are preserved exactly, which is essential for rigid-body motion.
- **Integration with graphics pipelines.** Rotation matrices fit naturally into homogeneous transformation matrices, allowing rotation, translation, scaling, and projection to be handled in a single framework.

- **Widespread support.** Nearly all graphics APIs, physics engines, and math libraries natively support matrix-based transformations.

Despite these advantages, problems arise when rotations are parameterized using angles.

5.2 Axis-Aligned Rotations and Euler Angles

In practice, rotations are often constructed by composing rotations about the coordinate axes. Let

$$R_x(\alpha), \quad R_y(\beta), \quad R_z(\gamma)$$

denote rotations about the x-, y-, and z-axes. A common convention is

$$R = R_z(\gamma)\, R_y(\beta)\, R_x(\alpha),$$

This representation is intuitive as each angle corresponds to a familiar motion. However, it ties orientation to a fixed rotation order and to specific axes. In this specific example we are first rotating by α about the x-axis, then by β about the y-axis, and finally by γ about the z-axis. This happens because matrix multiplication is not commutative, so changing the order of multiplying these matrices will result in changes with the final rotation result.

5.3 What Gimbal Lock Is (Geometric View)

Gimbal lock occurs when two of the rotation axes become aligned. When this happens, one rotational degree of freedom is lost. This happens when two independent angle changes produce the same physical rotation.

A classic mechanical interpretation uses three nested gimbals. When the middle gimbal rotates to $\pm 90°$, the inner and outer gimbals line up, leaving only two independent axes of rotation.

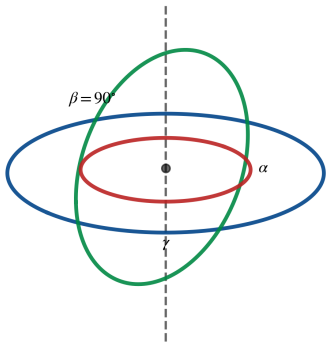

Figure 6: Physical interpretation of gimbal lock. When the middle gimbal reaches $\pm 90°$, two rotation axes align and one degree of freedom is lost. Figure generated by the author using custom Python code from [9].

This example shows that gimbal lock is a geometric issue, not a numerical one.

5.4 Gimbal Lock (Mathematical View)

Consider the axis-aligned rotation

$$R = R_z(\gamma)\, R_y(\beta)\, R_x(\alpha),$$

where

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}, \quad R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}, \quad R_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

At the singular angle

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

the middle rotation becomes

$$R_y\left(\frac{\pi}{2}\right) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}.$$

Substituting into the product gives

$$R = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}.$$

Multiplying the first two matrices yields

$$= \begin{pmatrix} 0 & -\sin\gamma & \cos\gamma \\ 0 & \cos\gamma & \sin\gamma \\ -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}.$$

Carrying out the final multiplication gives

$$R = \begin{pmatrix} 0 & \sin(\alpha-\gamma) & \cos(\alpha-\gamma) \\ 0 & \cos(\alpha-\gamma) & -\sin(\alpha-\gamma) \\ -1 & 0 & 0 \end{pmatrix}.$$

Key observation. The resulting matrix depends only on the *difference*

$$\alpha - \gamma,$$

not on α and γ independently. That is,

$$R(\alpha, \gamma) = R(\alpha - \gamma).$$

Consequences.

- One rotational degree of freedom is lost.
- Yaw and roll become mathematically dependent.
- Two rotation axes align, leaving only one effective axis.

This collapse is the precise mathematical expression of gimbal lock.

5.5 Why This Is a Problem in Practice

Near a gimbal lock configuration, small changes in input angles can cause large or unintuitive changes in orientation. Controls may feel unstable, camera motion can become unpredictable, and interpolation between orientations can behave poorly.

Importantly, switching to a different Euler angle convention does not fix the problem. It only moves the singularity to a different configuration.

Additional limitations appear when rotation matrices are constructed and manipulated through Euler angles:

- **Inefficient representation.** A 3D rotation has three degrees of freedom, but a rotation matrix stores nine numbers. The extra constraints must be maintained explicitly.
- **Non-unique angle decomposition.** Multiple Euler angle triples can represent the same rotation matrix, making it impossible to recover a unique set of angles.
- **Interpolation breaks geometry.** Linearly interpolating matrix entries does not preserve orthogonality, leading to distortion unless re-orthogonalization is applied.
- **Opaque composition.** Multiplying rotation matrices hides the underlying rotation axis and angle, making combined rotations difficult to interpret.
- **Unavoidable singularities.** Gimbal lock arises from the axis-based parameterization itself. No Euler convention removes the problem; it only shifts where it occurs.

5.6 Why We Move Beyond Rotation Matrices

Rotation matrices themselves are not flawed. The core issue lies in how rotations are parameterized and interpolated using angles. Because gimbal lock and non-uniqueness are structural problems of Euler-angle representations, they cannot be fixed by better numerics or different conventions. This motivates the use of rotation representations that encode rotation more directly.

6 Euler's Rotation Theorem

References. This section is based on and informed by the works of [6], [7].

Up to this point, we have described rotations using Euler angles and rotation matrices. While this approach is mathematically correct, it introduces ambiguity, redundancy, and unintuitive behavior that makes it difficult to work with in practice, especially for animation and interpolation. This leads to a natural question: is there a more direct way to describe what a rotation actually is?

Euler's Rotation Theorem provides this second way of representing rotations.

Euler originally stated this result in Latin in his work on rigid body motion. When translated into English, the theorem can be stated as follows:

> *No matter how a sphere is rotated about its center, a diameter can always be assigned whose direction in the translated position agrees with the initial position.*

What Euler meant by this is that for any rotation, there exists a direction in space that remains fixed under the rotation. This fixed direction defines the axis of rotation. Once this axis is identified, the entire rotation can be described as a single turn by some angle θ about that axis.

This immediately tells us something very important: to describe a rotation in three dimensions, we do not need three separate rotations or a full matrix. All we need is an axis of rotation and an angle of rotation. The axis is a unit vector which requires three numbers to describe, and the angle θ, which is a single scalar. Together, this means that a rotation can be fully described using exactly four numbers: one angle and three numbers specifying the axis.

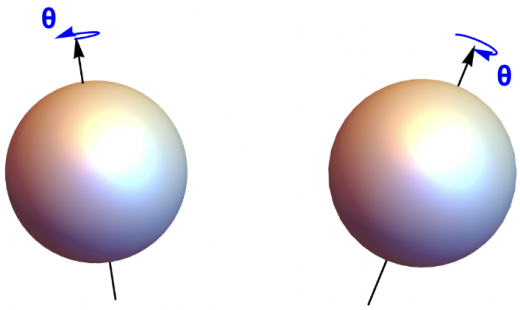

Figure 7: Euler angles as sequential rotations about coordinate axes. The black line indicates the rotation axis and the blue arrow labeled θ denotes the rotation angle. Figure generated by the author using custom Mathematica code from [9].

So how can Euler's Rotation Theorem be used to represent rotations using a structure better suited than rotation matrices? One such structure is the quaternion.

7 Quaternions

References. This section is based on and informed by the works of [2],[6],[8].

Quaternions were invented by Sir William Rowan Hamilton in 1843. Just as complex numbers provide a natural and compact way to represent rotation in two dimensions, Hamilton's goal was to generalize complex numbers so that they could represent rotation in three dimensions.

While it is not possible to construct a closed multiplication for three-dimensional complex numbers, Hamilton discovered that a four-dimensional extension admits such a structure. This led to the definition of quaternions, which extend complex numbers by introducing three imaginary units, traditionally denoted i, j, and k, satisfying

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

A quaternion can be written algebraically as

$$q = s + ix + jy + kz,$$

where $s, x, y, z \in \mathbb{R}$. Equivalently, a quaternion can be viewed geometrically as a scalar–vector pair

$$q = [s, \mathbf{v}],$$

where $s \in \mathbb{R}$ and $\mathbf{v} \in \mathbb{R}^3$.

This decomposition connects directly to Euler's Rotation Theorem. A three-dimensional rotation consists of an axis and an angle, and these are encoded in the vector and scalar parts of a quaternion, respectively. When restricted to unit length, quaternions represent pure rotations without scaling.

A unit quaternion lies on the three-sphere. Unlike the 1-sphere and the 2-sphere, we cannot directly visualize the 3-sphere, because it exists in four dimensions. Initially this can be frustrating, especially from an artistic perspective. We want mathematics to clearly explain what is happening in three-dimensional space, so how are quaternions supposed to explain rotation if we cannot even visually comprehend the space they live in?

7.1 How to Visualize S^3: Hopf Fibrations

An answer can be derived using the work of Heinz Hopf. Hopf used stereographic projection to map the three-sphere into three-dimensional space. Instead of mapping each point individually, he showed that each point on the 2-sphere corresponds to an entire circle in the three-sphere. Each of these circles is a geodesic of the three-sphere.

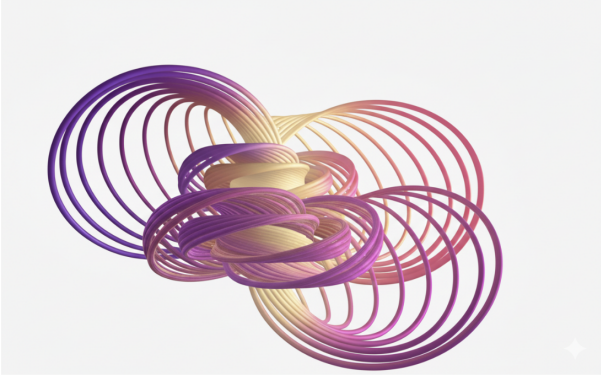

Figure 8: Google. Hopf Fibration Image. Gemini, [29 Nov. 2025]. AI-generated image.

The result of this construction is an extremely abstract-looking image (figure 8), which Hopf called the Hopf fibration. At first glance, the Hopf fibration does not seem very helpful for understanding quaternions. It appears to be nothing more than a collection of interlocked circles. So what does this image actually represent?

Each of these circles is called a Hopf fiber. A single point on a Hopf fiber corresponds to a single quaternion, while the entire Hopf fiber represents all unit quaternions that produce the same rotation axis.

7.2 Interpreting the Hopf Fibration

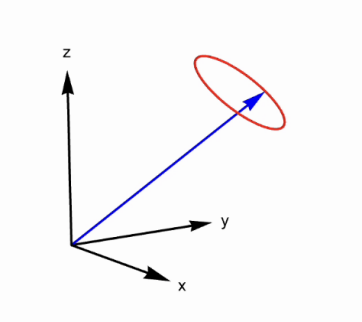

Figure 9: The arrow rotating along the path of the red circle, illustrating a rotation about a fixed axis. Figure generated by the author using custom Mathematica code from [9].

To see how this relates to rotations in the three-dimensional world, consider this second image in Figure 8. In this image, an arrow traces out a circular path. You can think of this arrow as the axis of rotation, or more intuitively, as your arm. If you extend your arm straight out and rotate it through a full circle, your hand traces out a circular path in space. That circular path is a Hopf fiber. Each position of your hand along that path corresponds to a unit quaternion.

With this interpretation, quaternions stop feeling like abstract objects from four-dimensional

space and instead become concrete descriptions of rotations that we can physically imagine.

8 Quaternion Algebra

References. This section is based on and informed by the works of [2],[6].

A quaternion is an extension of complex numbers and can be written as

$$q = [s, \mathbf{v}] = [s, (x, y, z)],$$

where $s \in \mathbb{R}$ is the scalar part and $\mathbf{v} \in \mathbb{R}^3$ is the vector part. Equivalently, a quaternion may be viewed as a point in four-dimensional space $\mathbb{R}^4$.

8.1 Addition and Subtraction

Let $q = [s, \mathbf{v}]$ and $q' = [s', \mathbf{v}']$ be two quaternions. Addition is defined by adding the scalar and vector parts independently:

$$q + q' = [\, s + s',\ \mathbf{v} + \mathbf{v}' \,].$$

Subtraction is defined in the same component-wise manner:

$$q - q' = [\, s - s',\ \mathbf{v} - \mathbf{v}' \,].$$

These operations behave exactly like vector addition and subtraction in $\mathbb{R}^4$. However, addition and subtraction do not correspond to composing rotations and are therefore not used to combine rotational motions.

8.2 Scalar Multiplication

If $r \in \mathbb{R}$ is a real number and $q = [s, \mathbf{v}]$, scalar multiplication is defined as

$$rq = [rs, r\mathbf{v}].$$

Scalar multiplication changes the magnitude of a quaternion but does not preserve rotations unless the quaternion is renormalized.

8.3 Quaternion Multiplication

Quaternion multiplication is defined differently from addition. Given two quaternions $q = [s, \mathbf{v}]$ and $q' = [s', \mathbf{v}']$, their product is

$$qq' = \big[ss' - \mathbf{v} \cdot \mathbf{v}',\ \mathbf{v} \times \mathbf{v}' + s\mathbf{v}' + s'\mathbf{v}\big].$$

Quaternion multiplication is not commutative:

$$qq' \neq q'q.$$

This is because quaternion multiplication corresponds to the composition of rotations. If q_1 and q_2 represent rotations, then the product $q_2 q_1$ represents the rotation obtained by first applying q_1 and then applying q_2. Since the order of rotations matters, quaternion multiplication is not commutative.

However, quaternion multiplication is associative and distributes over addition.

8.4 Conjugate and Norm

The conjugate of a quaternion $q = [s, \mathbf{v}]$ is defined as

$$q^* = [s, -\mathbf{v}].$$

The norm of a quaternion is defined by

$$\|q\| = \sqrt{s^2 + x^2 + y^2 + z^2}.$$

The norm measures the distance of a quaternion from the origin in $\mathbb{R}^4$.

8.5 Unit Quaternions and the Three-Sphere

A quaternion is called a *unit quaternion* if its norm is equal to one:

$$\|q\| = 1.$$

The set of all unit quaternions satisfies

$$s^2 + x^2 + y^2 + z^2 = 1,$$

which is exactly the equation of the three-sphere S^3 embedded in $\mathbb{R}^4$.

Thus, while all quaternions live in four-dimensional space, only unit quaternions lie on the surface of the three-sphere. Quaternions with norm less than one lie inside S^3, and those with norm greater than one lie outside it.

Only unit quaternions represent valid rotations.

8.6 Normalization

Any nonzero quaternion can be converted into a unit quaternion by normalization:

$$\hat{q} = \frac{q}{\|q\|}.$$

In component form, this becomes

$$\hat{q} = \left[\frac{s}{\sqrt{s^2 + x^2 + y^2 + z^2}}, \ \frac{(x, y, z)}{\sqrt{s^2 + x^2 + y^2 + z^2}} \right].$$

Normalization is necessary because only unit quaternions lie on the three-sphere S^3. If a quaternion is not normalized, it does not represent a pure rotation.

8.7 Why Normalization Matters for Rotations

Quaternion based rotation of a vector is performed using the formula

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

where $p = [0, \mathbf{v}]$ is the pure quaternion representing a vector in $\mathbb{R}^3$, and q is the quaternion encoding the rotation.

For this formula to represent a valid rotation, the quaternion q must be a unit quaternion. In general, the inverse of a quaternion is given by

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

However, when q is a unit quaternion, its norm satisfies $\|q\| = 1$, and the inverse simplifies to

$$q^{-1} = q^*.$$

This simplification is not just convenient; it is essential. If q is not normalized, then the factor $\|q\|^2$ appears in the denominator of the inverse. As a result, the transformation

$$q\,p\,q^{-1}$$

introduces an unwanted scaling effect in addition to rotation.

Geometrically, only unit quaternions lie on the three-sphere S^3, which is the space of valid rotations. If a quaternion drifts off S^3, it no longer represents a pure rotation.

In practice, numerical operations such as repeated multiplication or interpolation can cause small errors that change the norm of a quaternion. Normalization corrects these errors by projecting the quaternion back onto the unit three-sphere, ensuring that the rotation formula continues to represent a pure rotation.

8.8 Symmetry of q and $-q$

A unit quaternion q and its negative $-q$ represent the same spatial rotation. This follows from the fact that rotations are applied to a vector $v \in \mathbb{R}^3$ via the mapping

$$v \mapsto q\,v\,q^{-1},$$

and replacing q with $-q$ leaves the result unchanged:

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

As a result, the space of unit quaternions double-covers the space of three-dimensional rotations, meaning each rotation in SO(3) corresponds to exactly two unit quaternions, q and $-q$

9 Exp and Log Maps as Bridges Between $\mathbb{R}^3$ and S^3

References. This section is based on and informed by the works of [2],[4].

The exponential and logarithm maps can be understood as tools that translate between two different ways of representing rotation data.

Vectors in $\mathbb{R}^3$ provide a convenient, linear way to describe rotations using axis–angle form. However, the space of rotations itself is not linear. Instead, it is represented by unit quaternions living on the curved three-sphere S^3 embedded in four-dimensional space.

The exponential map takes rotation data expressed in $\mathbb{R}^3$ and lifts it into four dimensions:

$$\exp : \mathbb{R}^3 \longrightarrow S^3 \subset \mathbb{R}^4.$$

Let

$$q = [0, \theta \mathbf{v}],$$

where $\theta \in \mathbb{R}$ and $\mathbf{v} \in \mathbb{R}^3$ is a unit vector. The exponential map is given by

$$\exp(q) = [\cos\theta, \sin\theta\, \mathbf{v}].$$

Geometrically, this process takes a direction and magnitude in three-dimensional space (an axis and an angle) and moves along a great circle on S^3 starting from the identity quaternion. The result is a unit quaternion that encodes the same rotation, now represented as a point on the three-sphere. In this sense, the exponential map converts linear rotation data into a curved, four-dimensional representation that respects the geometry of rotation.

The logarithm map performs the inverse operation:

$$\log : S^3 \subset \mathbb{R}^4 \longrightarrow \mathbb{R}^3.$$

Let

$$q = [\cos\theta, \sin\theta\, \mathbf{v}]$$

be a unit quaternion. The logarithm map is defined by

$$\log(q) = [0, \theta \mathbf{v}].$$

Given a unit quaternion representing a rotation, the logarithm map projects this four-dimensional information back down into three dimensions. It extracts the rotation axis and angle encoded by the quaternion and returns them as a vector in $\mathbb{R}^3$.

While rotations live naturally on the curved space S^3, the logarithm map allows us to work with them in a linear space where standard operations such as interpolation, averaging, and differentiation are easier to define.

Together, the exponential and logarithm maps form a bridge between three-dimensional rotation data and its four-dimensional quaternion representation. This is why they play a central role in interpolation methods and in the analysis of rotational motion.

10 Benefits of Quaternions

References. This section is based on and informed by the works of [2],[6].

Quaternions provide a robust and efficient way to represent rotations in three dimensions. Many of their advantages come from the fact that unit quaternions lie on the three-sphere S^3, which naturally encodes the geometry of rotation.

- **No gimbal lock.** Quaternions avoid gimbal lock because rotations are not represented as sequential rotations about coordinate axes.
- **Compact representation.** A quaternion represents a rotation using only four numbers, compared to nine numbers in a 3×3 rotation matrix, with no redundant information.
- **Easy to keep valid.** A rotation quaternion remains valid as long as it has unit length. Numerical drift can be corrected simply by normalization.
- **Numerical stability.** Quaternions are more stable than rotation matrices for long animation chains and repeated updates, avoiding gradual deformation and drift.
- **Efficient inversion.** The inverse of a unit quaternion is given by its conjugate, making inverse rotations computationally inexpensive.
- **Smooth interpolation.** Because unit quaternions lie on S^3, interpolation corresponds to moving along geodesics on the three-sphere, producing smooth, shortest-path rotations with constant angular velocity.

11 Limitations of Quaternions

References. This section is based on and informed by the works of [2],[6].

While quaternions are extremely effective for representing rotation, they are not a complete solution for all transformation tasks in computer graphics and game engines.

- **Rotation only.** Although quaternion-based extensions exist, combining rotation and translation into a single quaternion-based framework is more complex than using homogeneous matrices (matrices that combine rotation with translation) and is rarely done in practice.
- **Steeper learning curve.** Quaternion algebra is non-commutative and operates in four dimensions, making it less intuitive than matrix-based approaches for many users.
- **Difficult to visualize.** Because quaternions live in four-dimensional space, they cannot be directly visualized, requiring indirect interpretations such as axis–angle form or Hopf fibrations.

- **Double-cover ambiguity.** Each rotation is represented by two quaternions, q and $-q$. While mathematically natural, this requires care during interpolation to avoid discontinuities.

12 Geodesics on S^3

References. This section is based on and informed by the works of [1],[2].

Geodesics are the shortest paths between points on a curved surface. They generalize the idea of straight lines in Euclidean space to curved manifolds.

Just as:

- geodesics on S^2 are great circles in $\mathbb{R}^3$,
- geodesics on S^3 are great circles in $\mathbb{R}^4$,

geodesics on the three-sphere represent the most direct paths between unit quaternions.

Because unit quaternions lie on S^3, any smooth transition between two rotations must follow a curve on this manifold. The shortest such transition is a geodesic.

12.1 Why Geodesics Matter for Rotations

Geodesics are fundamental to rotation interpolation. Whenever we interpolate between two orientations, we are implicitly choosing a path between two points on S^3.

Following a geodesic ensures that:

- the rotation follows the shortest path between orientations,
- the angular velocity remains constant,
- no unnecessary twisting or distortion is introduced.

This is why interpolation methods such as spherical linear interpolation (SLERP) are defined using geodesics on S^3. Rather than interpolating coordinates independently, SLERP moves along a great circle on the three-sphere, respecting the intrinsic geometry of rotation space.

From a differential geometry perspective, this highlights a key idea that rotations do not live in a flat space. Treating them as points on a curved manifold and interpolating along geodesics is essential for producing smooth and physically meaningful motion in animation, simulation, and game engines.

13 SnapTurn: Minimal Rotation Between Two Directions

References. This section is based on and informed by the works of [2],[6].

13.1 Motivation

After computing where an object should move using `ShortestCurve()` and how its orientation evolves along a surface using `CarryNoTwist()`, we still need a way to rotate an object directly from one direction to another. This happens often in practice: snapping a camera to face a target, turning a character toward a goal, or aligning an object with a surface normal.

The goal of `SnapTurn()` is to perform this rotation in the cleanest possible way. In particular, we want a rotation that is correct, stable, and introduces no unnecessary twisting. This section constructs such a rotation using quaternions and proves that it is the unique minimal rotation taking one direction to another.

13.2 What We Want to Prove

Let $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ be unit vectors with $\mathbf{a} \neq -\mathbf{b}$. Define

$$d = \mathbf{a} \cdot \mathbf{b}, \qquad \mathbf{c} = \mathbf{a} \times \mathbf{b}.$$

Consider the quaternion

$$q = \frac{(1 + d,\ \mathbf{c})}{\|(1 + d,\ \mathbf{c})\|}.$$

To show that this quaternion defines the correct minimal rotation, we prove the following four properties:

1. **(Unit quaternion)** q has unit length.
2. **(Correct rotation)** The induced rotation satisfies $R_q(\mathbf{a}) = \mathbf{b}$.
3. **(Minimal rotation angle)** The rotation angle of q equals the smaller angle between $\mathbf{a}$ and $\mathbf{b}$.
4. **(Minimal-norm axis)** The rotation axis is $\dfrac{\mathbf{a} \times \mathbf{b}}{\|\mathbf{a} \times \mathbf{b}\|}$, which is perpendicular to both directions.

Together, these show that q is the unique minimal rotation sending $\mathbf{a}$ to $\mathbf{b}$.

13.3 Proof 1: Unit Length

Since $\mathbf{a}$ and $\mathbf{b}$ are unit vectors,

$$\|\mathbf{a}\| = \|\mathbf{b}\| = 1.$$

Using the standard identity

$$\|\mathbf{a} \times \mathbf{b}\|^2 = \|\mathbf{a}\|^2\|\mathbf{b}\|^2 - (\mathbf{a} \cdot \mathbf{b})^2,$$

we obtain

$$\|\mathbf{c}\|^2 = 1 - d^2.$$

Now compute the squared norm of the raw quaternion:

$$\begin{aligned}\|(1+d,\ \mathbf{c})\|^2 &= (1+d)^2 + \|\mathbf{c}\|^2 \\ &= (1+d)^2 + (1-d^2) \\ &= 1 + 2d + d^2 + 1 - d^2 \\ &= 2(1+d).\end{aligned}$$

Normalizing by $\sqrt{2(1+d)}$ gives a unit quaternion.

Quaternion Rotation Formula (Explicit Derivation)

Let $q = (w, \mathbf{u})$ be a unit quaternion, so that

$$q^{-1} = (w, -\mathbf{u}).$$

A vector $\mathbf{x} \in \mathbb{R}^3$ is identified with the pure quaternion $(0, \mathbf{x})$. The rotation induced by q is defined as

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

Step 1: First multiplication. Using the quaternion multiplication rule

$$(s, \mathbf{v})(t, \mathbf{w}) = (st - \mathbf{v} \cdot \mathbf{w},\ s\mathbf{w} + t\mathbf{v} + \mathbf{v} \times \mathbf{w}),$$

we compute

$$q(0, \mathbf{x}) = (w, \mathbf{u})(0, \mathbf{x}) = (-\mathbf{u} \cdot \mathbf{x},\ w\mathbf{x} + \mathbf{u} \times \mathbf{x}).$$

Denote this intermediate quaternion by $(\alpha, \boldsymbol{\beta})$, where

$$\alpha = -\mathbf{u} \cdot \mathbf{x}, \qquad \boldsymbol{\beta} = w\mathbf{x} + \mathbf{u} \times \mathbf{x}.$$

Step 2: Multiply by q^{-1}. We now compute

$$(\alpha, \boldsymbol{\beta})(w, -\mathbf{u}) = (\alpha w - \boldsymbol{\beta} \cdot (-\mathbf{u}),\ \alpha(-\mathbf{u}) + w\boldsymbol{\beta} + \boldsymbol{\beta} \times (-\mathbf{u})).$$

We are interested only in the vector part.

Step 3: Expand each vector term. First,

$$\alpha(-\mathbf{u}) = -(\mathbf{u}\cdot\mathbf{x})\mathbf{u}.$$

Second,

$$w\boldsymbol{\beta} = w(w\mathbf{x} + \mathbf{u}\times\mathbf{x}) = w^2\mathbf{x} + w(\mathbf{u}\times\mathbf{x}).$$

Third,

$$\begin{aligned}\boldsymbol{\beta}\times(-\mathbf{u}) &= -(w\mathbf{x} + \mathbf{u}\times\mathbf{x})\times\mathbf{u} \\ &= -\big(w(\mathbf{x}\times\mathbf{u}) + (\mathbf{u}\times\mathbf{x})\times\mathbf{u}\big).\end{aligned}$$

Using $\mathbf{x}\times\mathbf{u} = -\mathbf{u}\times\mathbf{x}$ and the vector triple-product identity

$$(\mathbf{u}\times\mathbf{x})\times\mathbf{u} = \mathbf{x}\|\mathbf{u}\|^2 - (\mathbf{u}\cdot\mathbf{x})\mathbf{u},$$

we obtain

$$\boldsymbol{\beta}\times(-\mathbf{u}) = w(\mathbf{u}\times\mathbf{x}) - \mathbf{x}\|\mathbf{u}\|^2 + (\mathbf{u}\cdot\mathbf{x})\mathbf{u}.$$

Step 4: Combine all vector terms. Adding the three contributions gives

$$R_q(\mathbf{x}) = \big(w^2 - \|\mathbf{u}\|^2\big)\mathbf{x} + 2(\mathbf{u}\cdot\mathbf{x})\mathbf{u} + 2w(\mathbf{u}\times\mathbf{x}).$$

Conclusion. The action of a unit quaternion on vectors can be written entirely in terms of dot and cross products. This explicit formula allows us to verify exactly how the SnapTurn quaternion rotates directions.

13.4 Proof 2: Correct Rotation from a to b

For the SnapTurn quaternion,

$$w = \sqrt{\frac{1+d}{2}}, \qquad \mathbf{u} = \frac{\mathbf{c}}{\sqrt{2(1+d)}}.$$

Step 1: Compute $w^2 - \|\mathbf{u}\|^2$. Since $\|\mathbf{c}\|^2 = 1 - d^2$,

$$\|\mathbf{u}\|^2 = \frac{1-d^2}{2(1+d)} = \frac{1-d}{2}.$$

Thus,

$$w^2 - \|\mathbf{u}\|^2 = \frac{1+d}{2} - \frac{1-d}{2} = d.$$

Step 2: Dot-product term. Because $\mathbf{c} = \mathbf{a}\times\mathbf{b}$ is perpendicular to $\mathbf{a}$,

$$\mathbf{u}\cdot\mathbf{a} = 0.$$

Step 3: Cross-product term. Using the identity

$$(\mathbf{a} \times \mathbf{b}) \times \mathbf{a} = \mathbf{b} - d\mathbf{a},$$

we compute

$$2w(\mathbf{u} \times \mathbf{a}) = \mathbf{b} - d\mathbf{a}.$$

Step 4: Combine terms.

$$R_q(\mathbf{a}) = d\mathbf{a} + 0 + (\mathbf{b} - d\mathbf{a}) = \mathbf{b}.$$

Conclusion. The SnapTurn quaternion rotates $\mathbf{a}$ exactly into $\mathbf{b}$.

13.5 Proof 3: Axis of Rotation

Any unit quaternion $(w, \mathbf{u})$ has axis–angle form

$$(w, \mathbf{u}) = \left(\cos\frac{\varphi}{2},\ \hat{\mathbf{n}} \sin\frac{\varphi}{2}\right),$$

where $\hat{\mathbf{n}} = \mathbf{u}/\|\mathbf{u}\|$ is the rotation axis.

Since $\mathbf{u}$ is proportional to $\mathbf{a} \times \mathbf{b}$, the axis is

$$\hat{\mathbf{n}} = \frac{\mathbf{a} \times \mathbf{b}}{\|\mathbf{a} \times \mathbf{b}\|}.$$

Conclusion. The rotation axis is perpendicular to both $\mathbf{a}$ and $\mathbf{b}$ and corresponds to the shortest possible rotation axis.

Why We Prove Minimality

So far, we have shown that the SnapTurn quaternion is valid and maps $\mathbf{a}$ to $\mathbf{b}$. For future use, such as interpolation, composing rotations, or correcting orientation drift, it is important that this rotation be minimal. Larger rotations introduce unnecessary twisting and can accumulate errors. We therefore explicitly prove that SnapTurn achieves the smallest possible rotation angle.

13.6 Proof 4: The Rotation Angle Is Minimal

Let θ be the angle between $\mathbf{a}$ and $\mathbf{b}$, so that

$$\cos\theta = \mathbf{a} \cdot \mathbf{b}.$$

Step 1: Angle of the SnapTurn quaternion. The scalar part of q is

$$w = \sqrt{\frac{1 + \mathbf{a} \cdot \mathbf{b}}{2}} = \cos\frac{\theta}{2}.$$

For a unit quaternion, $w = \cos(\varphi/2)$, where φ is the rotation angle. Hence,

$$\varphi = 2\arccos(w) = \theta.$$

Step 2: Minimality argument. Let R be any rotation sending $\mathbf{a}$ to $\mathbf{b}$, with axis $\hat{\mathbf{n}}$ and angle ϕ. A standard rotation identity gives

$$\mathbf{a} \cdot \mathbf{b} = (\hat{\mathbf{n}} \cdot \mathbf{a})^2 + \left(1 - (\hat{\mathbf{n}} \cdot \mathbf{a})^2\right) \cos\phi.$$

Solving for $\cos\phi$ yields

$$\cos\phi = \frac{\cos\theta - (\hat{\mathbf{n}} \cdot \mathbf{a})^2}{1 - (\hat{\mathbf{n}} \cdot \mathbf{a})^2} \leq \cos\theta,$$

with equality if and only if $\hat{\mathbf{n}} \cdot \mathbf{a} = 0$.

Since cosine is decreasing on $[0, \pi]$, this implies

$$\phi \geq \theta.$$

Conclusion. The smallest possible rotation angle mapping $\mathbf{a}$ to $\mathbf{b}$ is θ, achieved exactly when the rotation axis is parallel to $\mathbf{a} \times \mathbf{b}$. Thus, SnapTurn realizes the unique minimal rotation.

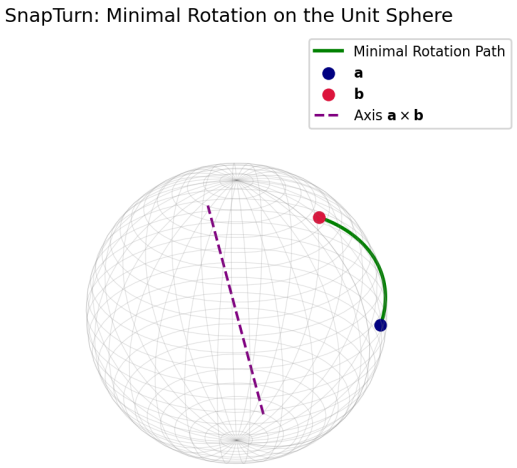

Figure 10: SnapTurn rotation between unit vectors $\mathbf{a}$ and $\mathbf{b}$ on the unit sphere. The green arc shows the minimal rotation (a great circle) in the plane spanned by $\mathbf{a}$ and $\mathbf{b}$. The rotation axis $\mathbf{a} \times \mathbf{b}$ is orthogonal to this plane. Figure generated by the author using custom Python code from [9].

13.7 Degenerate Case: $\mathbf{a} = -\mathbf{b}$

If $\mathbf{a} = -\mathbf{b}$, then

$$\mathbf{a} \cdot \mathbf{b} = -1, \qquad \mathbf{a} \times \mathbf{b} = \mathbf{0}.$$

In this situation, the SnapTurn formula

$$q = \frac{(1 + \mathbf{a} \cdot \mathbf{b},\ \mathbf{a} \times \mathbf{b})}{\sqrt{2(1 + \mathbf{a} \cdot \mathbf{b})}}$$

breaks down because both the numerator and denominator vanish.

Geometrically, this case corresponds to rotating a vector by 180°. Unlike the non-degenerate case, a 180° rotation does not have a unique rotation axis. Any axis perpendicular to $\mathbf{a}$ produces a valid rotation sending $\mathbf{a}$ to $-\mathbf{a}$.

This non-uniqueness explains why the SnapTurn construction excludes this case: there is no single minimal rotation axis to prefer. In practice, implementations handle this case separately by choosing an arbitrary axis perpendicular to $\mathbf{a}$, often using a fixed reference vector to ensure consistency across frames.

Interpretation. This figure visualizes the geometric content of the SnapTurn construction: the rotation follows the unique shortest path on S^2 and uses the only axis that produces the minimal rotation angle.

Conclusion. When $\mathbf{a} = -\mathbf{b}$, the minimal rotation angle is still well-defined (π), but the rotation axis is not. The degenerate case admits infinitely many minimal rotations, and SnapTurn must resolve this ambiguity explicitly.

14 Approximating Linear Interpolation on S^3

References. This section is based on and informed by the works of [2],[6].

14.1 Motivation

In Euclidean space, linear interpolation between two points is defined by the straight line connecting them. For points $p, r \in \mathbb{R}^n$, this interpolation is given by

$$\text{lerp}(p, r, t) = (1 - t)p + tr, \quad t \in [0, 1].$$

When working with rotations, however, the situation is more subtle. Unit quaternions represent rotations as points on the three-sphere $S^3 \subset \mathbb{R}^4$. The shortest and most natural interpolation between two rotations follows a geodesic (a great circle) on S^3, which is computed exactly using spherical linear interpolation (SLERP).

While SLERP produces geometrically correct results, it is computationally expensive and involves trigonometric functions. In applications such as inverse kinematics, optimization, and differentiation-based methods, it is often desirable to use a faster, simpler approximation. This motivates the construction of an approximate linear interpolation scheme that remains close to the geodesic while being easier to compute.

14.2 Setup

Let

$$q_A, q_B \in S^3$$

be two unit quaternions representing orientations. Since q and $-q$ encode the same physical rotation, we first enforce a short-arc condition to ensure interpolation follows the shorter geodesic on S^3:

$$\tilde{q}_B = \begin{cases} q_B, & \langle q_A, q_B \rangle \geq 0, \\ -q_B, & \langle q_A, q_B \rangle < 0. \end{cases}$$

This guarantees that the angle between q_A and $\tilde{q}_B$ is at most $\pi/2$, avoiding unnecessary long rotations during interpolation.

14.3 Definition of Approximating Linear Interpolation

We first interpolate linearly in the ambient space $\mathbb{R}^4$:

$$r(t) = (1 - t)q_A + t\tilde{q}_B, \quad t \in [0, 1].$$

In general, $r(t)$ is not a unit quaternion and therefore does not represent a valid rotation. To project back onto S^3, we normalize:

$$\text{ALI}(q_A, q_B, t) = \frac{r(t)}{\|r(t)\|}.$$

This procedure defines Approximating Linear Interpolation (ALI).

Property 1: ALI Produces a Unit Quaternion

Recall the definition:

$$\mathrm{ALI}(t) = \frac{(1-t)q_A + t\,\tilde{q}_B}{\|(1-t)q_A + t\,\tilde{q}_B\|}.$$

Let's call the numerator:

$$r(t) = (1-t)q_A + t\,\tilde{q}_B \in \mathbb{R}^4.$$

Now ALI is:

$$\mathrm{ALI}(t) = \frac{r(t)}{\|r(t)\|}.$$

Take the norm of both sides:

$$\|\,\mathrm{ALI}(t)\| = \left\| \frac{r(t)}{\|r(t)\|} \right\| = \frac{\|r(t)\|}{\|r(t)\|} = 1,$$

as long as $r(t) \neq 0$.

Conclusion. Only unit quaternions represent pure rotations. If interpolation were allowed to leave S^3, the result would include unwanted scaling or distortion. This property guarantees that ALI always produces a valid rotation that can be safely applied to objects in a game engine or animation system.

Property 2: Correct End Points

Claim. For $t = 0$ and $t = 1$,

$$\mathrm{ALI}(q_A, q_B, 0) = q_A, \qquad \mathrm{ALI}(q_A, q_B, 1) = \tilde{q}_B,$$

and $\tilde{q}_B$ represents the same physical rotation as q_B (they differ only by a global sign). Thus ALI goes from orientation A to orientation B along the short arc.

Proof for $t = 0$.

Recall:

$$r(t) = (1-t)q_A + t\tilde{q}_B, \qquad \mathrm{ALI}(t) = \frac{r(t)}{\|r(t)\|}.$$

Plug in $t = 0$:

$$r(0) = (1-0)q_A + 0 \cdot \tilde{q}_B = q_A.$$

Since q_A is a unit quaternion,

$$\|r(0)\| = \|q_A\| = 1.$$

Thus,

$$\mathrm{ALI}(0) = \frac{r(0)}{\|r(0)\|} = \frac{q_A}{1} = q_A.$$

So the interpolation starts exactly at q_A.

Proof for $t = 1$.

Plug in $t = 1$:

$$r(1) = (1-1)q_A + 1 \cdot \tilde{q}_B = \tilde{q}_B.$$

Again, $\tilde{q}_B$ is a unit quaternion, so $\|r(1)\| = 1$.

Thus,

$$\mathrm{ALI}(1) = \frac{r(1)}{\|r(1)\|} = \frac{\tilde{q}_B}{1} = \tilde{q}_B.$$

Recall: $\tilde{q}_B$ is either q_B or $-q_B$. Both represent the same 3D rotation. We choose the sign so that ALI follows the short arc, but physically the result is still orientation B.

Conclusion.

ALI starts at q_A and ends at q_B (via its shortest representative $\tilde{q}_B$), so it truly interpolates between the two orientations.

Property 3: ALI stays on the short arc between the two orientations

What we want to show

Because of the sign-fixing step:

$$\tilde{q}_B = \begin{cases} q_B, & \langle q_A, q_B \rangle \geq 0, \\ -q_B, & \langle q_A, q_B \rangle < 0, \end{cases}$$

we guarantee:

1. q_A and $\tilde{q}_B$ lie on the same hemisphere of S^3,
2. the interpolation never crosses the long way around the sphere,
3. the angle between q_A and $\tilde{q}_B$ is $\leq 90^\circ$ in $\mathbb{R}^4$, which corresponds to a rotation angle $\leq 180^\circ$ in 3D.

This ensures ALI always produces the *shortest rotation* between the two quaternions.

Fact. For any two unit vectors $u, v \in \mathbb{R}^n$:

- If $\langle u, v \rangle > 0$, the angle between them is *less than* 90°, so they lie in the *same hemisphere*.

- If $\langle u, v\rangle < 0$, the angle is *greater than* $90°$, so the shortest way to match them is to flip one sign.

That is exactly why we do the sign fix.

Step 1: Angle between q_A and $\tilde{q}_B$ is at most $\pi/2$.

Let θ denote the angle between the two unit vectors $q_A, \tilde{q}_B \in S^3 \subset \mathbb{R}^4$. By the standard inner-product/angle relation in $\mathbb{R}^4$,

$$\langle q_A, \tilde{q}_B\rangle = \|q_A\|\,\|\tilde{q}_B\|\cos\theta.$$

Since q_A and $\tilde{q}_B$ are unit quaternions, $\|q_A\| = \|\tilde{q}_B\| = 1$, so this simplifies to

$$\langle q_A, \tilde{q}_B\rangle = \cos\theta, \qquad \theta \in [0, \pi].$$

By construction of the sign fix we have

$$\langle q_A, \tilde{q}_B\rangle \geq 0.$$

Therefore

$$\cos\theta \geq 0 \quad \Longrightarrow \quad \theta \in \left[0, \tfrac{\pi}{2}\right].$$

So q_A and $\tilde{q}_B$ lie in the same hemisphere and the geodesic (shortest great-circle arc) between them has length $\leq \pi/2$ in S^3.

Step 2: Every chord point $r(t)$ stays in the same half-space.

We now look at the straight chord joining q_A and $\tilde{q}_B$:

$$r(t) = (1-t)q_A + t\,\tilde{q}_B, \qquad t \in [0, 1].$$

Compute its inner product with q_A:

$$\begin{aligned}
\langle r(t), q_A\rangle &= \big\langle (1-t)q_A + t\,\tilde{q}_B,\ q_A\big\rangle \\
&= (1-t)\,\langle q_A, q_A\rangle \ +\ t\,\langle \tilde{q}_B, q_A\rangle \\
&= (1-t)\,\|q_A\|^2 + t\,\langle q_A, \tilde{q}_B\rangle \\
&= (1-t)\cdot 1 + t\,\langle q_A, \tilde{q}_B\rangle \\
&= 1 - t + t\,\langle q_A, \tilde{q}_B\rangle.
\end{aligned}$$

Using $\langle q_A, \tilde{q}_B\rangle \geq 0$ we get the estimate

$$\langle r(t), q_A\rangle \ \geq\ 1 - t + t\cdot 0 \ =\ 1 - t \ \geq\ 0$$

for all $t \in [0, 1]$. Thus every point $r(t)$ lies in the closed half-space on the same side of the hyperplane orthogonal to q_A.

Step 3: Normalization keeps us in the same hemisphere. For $t \in [0,1]$ we know (from Property 1) that $r(t) \neq 0$, so $\mathrm{ALI}(t)$ is well-defined. Then

$$\begin{aligned}\big\langle \mathrm{ALI}(t), q_A \big\rangle &= \Big\langle \frac{r(t)}{\|r(t)\|}, q_A \Big\rangle \\ &= \frac{1}{\|r(t)\|} \langle r(t), q_A \rangle.\end{aligned}$$

We already proved that $\langle r(t), q_A \rangle \geq 0$ and clearly $\|r(t)\| > 0$. Hence

$$\big\langle \mathrm{ALI}(t), q_A \big\rangle \geq 0 \quad \text{for all } t \in [0,1].$$

Geometrically, the condition $\langle x, q_A \rangle \geq 0$ for $x \in S^3$ describes the closed hemisphere of S^3 containing q_A. So the entire interpolated path $\mathrm{ALI}(t)$ stays inside this hemisphere and never crosses the equator into the antipodal side. Since the endpoints q_A and $\tilde{q}_B$ are at spherical distance $\theta \leq \pi/2$, the rotation they represent in SO(3) has angle $2\theta \leq \pi$ (the standard quaternion-to-rotation relation). Thus ALI always realizes a rotation of angle at most π, i.e. the *shortest* possible rotation between the two orientations.

Conclusion.

Without this condition, interpolation could travel the long way around S^3, causing unexpected spins or flips in animation. This property guarantees that ALI follows the shortest rotational path, preserving visual continuity and preventing sudden reversals.

Property 4: Minimal-Norm Projection onto S^3

Step 1: Reduce to a generic vector

Forget quaternions for a moment.

Take any nonzero vector $v \in \mathbb{R}^n$. We want to find the closest unit vector u (i.e. $\|u\| = 1$) to v.

Claim:

$$u^* = \frac{v}{\|v\|}$$

is the unique closest unit vector to v.

If we prove this for a general v, then for our quaternion case we simply set $v = r(t)$ and $u^* = \mathrm{ALI}(t)$.

Step 2: Use dot products

Take any unit vector u (so $\|u\| = 1$). Compute the squared distance to v:

$$\begin{aligned}\|v - u\|^2 &= \langle v - u,\, v - u \rangle \\ &= \|v\|^2 + \|u\|^2 - 2\langle v, u \rangle \\ &= \|v\|^2 + 1 - 2\langle v, u \rangle.\end{aligned}$$

Observe:

- $\|v\|^2$ is fixed,
- 1 is constant,
- so the only term that changes with u is $-2\langle v, u\rangle$.

Thus, to minimize $\|v - u\|^2$, we must *maximize* the dot product $\langle v, u\rangle$.

So:

"Closest unit vector to v" = "unit vector u with the largest dot product with v."

Step 3: Maximize the dot product

By Cauchy–Schwarz:

$$\langle v, u\rangle \leq \|v\| \cdot \|u\| = \|v\| \cdot 1 = \|v\|.$$

Equality occurs only if u is a positive scalar multiple of v:

$$u = \frac{v}{\|v\|}.$$

Thus:

$$\langle v, u\rangle \text{ is maximized exactly when } u = \frac{v}{\|v\|}.$$

Therefore:

$$\|v - u\|^2 \quad \text{is minimized at the same } u.$$

Hence the unique closest unit vector to v is:

$$u^* = \frac{v}{\|v\|}.$$

Step 4: Apply this to our case

Now take $v = r(t)$.

Then the closest unit vector to $r(t)$ is:

$$u^* = \frac{r(t)}{\|r(t)\|} = \mathrm{ALI}(t).$$

Thus for any other unit quaternion $q \in S^3$,

$$\|r(t) - \mathrm{ALI}(t)\| \ \leq \ \|r(t) - q\|.$$

Conclusion.

This shows that ALI does not arbitrarily modify the linear interpolation. Instead, it makes the smallest possible correction needed to return to S^3. As a result, ALI stays as close as possible to linear interpolation while still producing a valid rotation, explaining why it behaves smoothly and predictably in practice.

14.4 Interpretation

Approximating Linear Interpolation replaces the geodesic on S^3 with a straight chord in $\mathbb{R}^4$, followed by normalization. While this does not exactly follow the true geodesic, it preserves the correct endpoints, remains on the short arc, produces valid rotations, and minimizes deviation from the linear path. This makes ALI a practical compromise between geometric accuracy and computational efficiency, particularly in real-time systems where speed and differentiability are critical.

References

[1] **Crane, K.** (2025). *Discrete Differential Geometry: An Applied Introduction*. (Last updated: January 29, 2025).

[2] **Dam, E. B., Koch, M., & Lillholm, M.** (1998). *Quaternions, Interpolation and Animation* (Technical Report DIKU-TR-98/5). Department of Computer Science, University of Copenhagen.

[3] **do Carmo, M. P.** (1976). *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

[4] **Grassia, F. S.** (1998). Practical Parameterization of Rotations Using the Exponential Map. *The Journal of Graphics Tools*, vol. 3, no. 3, pp. 29–48.

[5] **Shifrin, T.** (2024). *Differential Geometry: A First Course in Curves and Surfaces* (AMS Open Math Notes). University of Georgia. (Last Revised: October 2, 2024).

[6] **Shoemake, K.** (1985). Animating Rotation with Quaternion Curves. *Computer Graphics* (Proceedings of SIGGRAPH '85), vol. 19, no. 3, pp. 245–254.

[7] **Wikipedia**. Euler's Rotation Theorem. Retrieved December 15, 2025, from `https://en.wikipedia.org/wiki/Euler%27s_rotation_theorem`.

[8] **Wikipedia**. Hopf Fibration. Retrieved December 15, 2025, from `https://en.wikipedia.org/wiki/Hopf_fibration`.

[9] **Saleh, B., & Magno, F.** (2025). *Geometric Operators on Manifolds*. GitHub repository. `https://github.com/franm08/geometric-operators-on-manifolds`