Representing Rotations and Orientations in Geometric Computing

Jehee Lee
Seoul National University [*]

Abstract

Geometric computing with three-dimensional rotations and orientations is a fundamental issue in three-dimensional computer graphics. Our approach was inspired by affine geometry and coordinate-free geometric programming. The basic idea of affine geometry is to make a distinction between points and vectors. The relation between orientations and rotations is analogous to the relation between points and vectors. Similarly to affine geometry, we argue that rotations and orientations should be represented differently in geometric computing. We show that, in three-dimensional space, unit quaternions (or rotation matrices) cannot parameterize rotations without singularity, but rotation vectors can. Conversely, rotation vectors cannot parameterize orientations without singularity, but unit quaternions (or rotation matrices) can. From these observations, we suggest that orientations should be represented by unit quaternions (or rotation matrices) and rotations should be represented by three-dimensional vectors. Furthermore, a set of operations are identified for combining rotations and orientations. Those operations are geometrically meaningful and independent of the choice of reference coordinate frames.
Keywords: Rotation and orientation, coordinate-free geometric programming, unit quaternion, rotation vector, axis-angle representation.

1 Introduction

Geometric computing with three-dimensional rotations and orientations is a fundamental issue in three-dimensional computer graphics. Rotation is circular movement, while the orientation of a rigid object refers to the state of the object being oriented. These two geometric concepts have conventionally been considered to be interchangeable because the orientation of an object can be represented as a rotation from the reference orientation given a coordinate frame.

The goal of this paper is to provide a useful perspective of understanding, representing, and manipulating 3-dimensional orientations and rotations for geometric computing. Our approach is inspired by affine geometry and coordinate-free geometric programming. The key idea of affine geometry is to make a distinction between points and vectors, and define operations for combining them. Based upon affine geometry, Goldman [5, 6] and DeRose [3, 4, 16] pioneered a method of writing graphics programs that are independent of the choice of reference coordinate frames. The study on geometric algebra pursues a similar goal with various geometric primitives rather than just vectors and points [12, 13].

There exists a strong analogy between the way that points are related to vectors and the way that orientations are related to rotations. Both points and orientations are geometric entities that describe states of geometric

[*]email:jehee@cse.snu.ac.kr

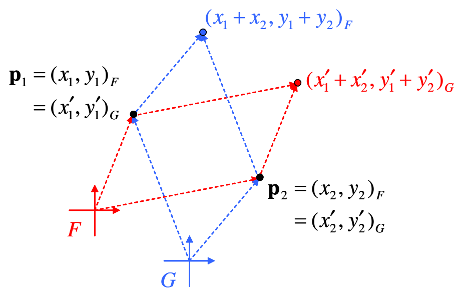
$(x_1 + x_2, y_1 + y_2)_F$
$\mathbf{p}_1 = (x_1, y_1)_F$
$= (x'_1, y'_1)_G$
$(x'_1 + x'_2, y'_1 + y'_2)_G$
$\mathbf{p}_2 = (x_2, y_2)_F$
$= (x'_2, y'_2)_G$
F
G

Figure 1: An example of coordinate-dependent operations. The element-wise addition of two points can actually be interpreted as the addition of two vectors that originate from the origin. The vector addition produces inconsistent results depending on where the origin is.

objects, while vectors and rotations are related to the change of states. Similarly to affine geometry, we found that a simple geometric algebra is available for representing and manipulating three-dimensional orientations and rotations. Central to our formalism is making a clear distinction between orientations and rotations, which are represented differently and appropriately for avoiding singularity in parameterization. We will show that, in three-dimensional space, unit quaternions (or rotation matrices) cannot parameterize rotations without singularity, but rotation vectors can. Conversely, rotation vectors cannot parameterize orientations without singularity, but unit quaternions (or rotation matrices) can. From these observations, we will argue that orientations should be represented by unit quaternions (or rotation matrices) and rotations should be represented by rotation vectors. Furthermore, we will identify a set of operations for combining them. Those operations are geometrically meaningful and coordinate-invariant. Since the algebraic structure and the representation power of rotation matrices are equivalent to those of unit quaternions, rotation matrices can be used instead of unit quaternions and vice versa (see [17] for details). So, we will not mention rotation matrices henceforth.

"Coordinate-free" does not imply that coordinates are unnecessary. We need to use coordinates in order to implement geometric algorithms. It is just that the geometric algorithms will produce consistent results independent of the choice of the reference frames in which the input data are represented. In that sense, "coordinate-invariance" might be an appropriate term rather than "coordinate-free". We use these two terms interchangeably throughout this paper.

The invariance on the choice of reference frames is of practical importance in producing the animation of rigid bodies and articulated figures because many existing animation systems have different conventions on choosing coordinate and joint reference frames. The operations between orientations and rotations are further generalized for the representation and manipulation of rigid body and articulated figure motions.

The main contribution of our work is the extension of the range of coordinate-invariant geometric programming by adding new entities and operations. The benefit of coordinate-invariant geometric programming is manifold. It provides us with geometric reasoning and intuition for the steps of our geometric programs. This allows us to write geometric programs relying on geometric reasoning rather than coordinate manipulations. It means that we do not need to fully understand all the details about quaternion-related mathematics for geometric programming.

2 Rotation and Orientation

Consider a rigid object in three-dimensional space. Affine geometry suggests that its position is represented by points, which can be distinguished from vectors. A point, as a geometric concept, is independent of the choice of a reference frame or an origin. Given a reference frame, a point can have its coordinates with respect to the origin. Vectors on the other hand have the attributes of direction and magnitude, but no fixed position. A vector can specify the relative movement of a point from one position to the other. With a reference frame fixed, we can represent any point as a vector that comes from the origin to the point. However, points and vectors are still different from a geometry point of view. An algebraic operation between points and vectors is geometrically meaningful if the result is independent of the choice of reference coordinate frames. For example, the sum of two vectors is well-defined algebraically and geometrically. However, the sum of two points is algebraically computable, but geometrically meaningless because the result depends on where the origin is (See Figure 1). Affine geometry defines a set of coordinate-invariant operations between points and vectors, which are summarized in Figure 4(left).

We observed that the relation between orientations and rotations is, in many aspects, analogous to the relation between points and vectors. From the geometry point of view, the orientation of a rigid object is independent of the choice of a reference orientation. Given a reference orientation, any orientation can be represented with respect to the reference orientation (see Figure 2(Left)). On the other hand, Euler's rotation theorem states that any rotation of a rigid object can be described by a fixed axis and an angle of rotation about the axis. The axis and the angle can be represented as a tuple of a unit vector $\hat{\mathbf{v}}$ and a scalar value θ. The rotation can also be represented by a single vector such that $\mathbf{v} = \theta\hat{\mathbf{v}} \in \mathbb{R}^3$. A single rotation vector is usually preferred over a separate axis-angle representation because the identity transform (zero rotation) is uniquely represented by the zero vector, while the identity transform can be represented by any combination of the zero angle and an arbitrary axis. The rotation have the attributes of direction and magnitude, but no fixed orientation. Rotation vectors can be arbitrarily long and rotation vectors longer than 2π correspond to multiple spinning of a rigid body.

With a reference frame fixed, we can represent any orientation of a rigid object as a relative rotation that brings the object aligned with the reference orientation to the target orientation. The relative rotational movement between any pair of two orientations (or coordinate systems) can be specified by a rotation vector(see Figure 2(Right)). However, in this case, the length of the vector is limited to interval $[0, 2\pi]$, which causes singularities in orientation parameterization. This observation has motivated the use of unit quaternions for the representation of three-dimensional orientations.

The mappings between orientations and rotations are singular in the sense that they have either many-to-one correspondences or discontinuous jumps. Therefore, orientations and rotations should be parameterized differently in order to avoid singularities. We will argue that we have to use vectors for parameterizing rotations and unit quaternions for orientations in three-dimensional space. We will discuss these singularity issues in the following sections: The two-dimensional cases in section 2.1 and the three-dimensional cases in section 2.2.

2.1 Two-dimensional space

Consider a rigid object that can rotate about the origin of the reference coordinate system in two-dimensional space. The rotation of a two-dimensional object has a single degree of freedom and therefore can be parameterized by scalar rotation angle θ. A positive angle denotes a counter-clockwise rotation and conversely

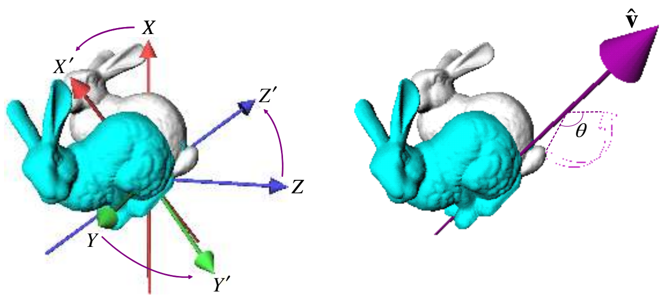

Figure 2: The reference coordinate system is depicted by the white bunny and XYZ axes. (Left) The orientation of the blue bunny can be described by a unit quaternion with respect to the reference coordinate system. $X'Y'Z'$ axes are a local coordinate system of the blue bunny. (Right) Euler's rotation theorem states that a fixed axis $\hat{\mathbf{v}}$ and an angle θ can always be found such that the rotation of the white bunny about axis $\hat{\mathbf{v}}$ by angle θ will bring it to the orientation of the blue bunny.

a negative angle denotes a clockwise rotation. A scalar value larger than 2π or smaller than -2π denotes multiple spinning of the object.

Parameterizing orientations. The orientation of an object can also be represented by scalar rotation angle from a given reference orientation, but many angles $\theta \pm 2n\pi$ for any integer n are mapped to a single orientation. To avoid many-to-one correspondences between orientations and rotation angles, the range of θ should be limited to a 2π-interval such as $[-\pi, \pi]$ or $[0, 2\pi]$. Consider a continuously-moving rigid object and its trajectory function $\theta(t)$ varying over time t. Confined within this 2π-interval, the trajectory of angle $\theta(t)$ may include discontinuous jumps from one boundary to the other even though the corresponding angular motion is continuous (see Figure 3 (left)). This problem can be avoided by using an extra parameter such that a point (x, y) on the unit circle represents an orientation. This representation is redundant in the sense that it uses more parameters than actually needed. The redundancy is compensated by the unit-length constraint $x^2 + y^2 = 1$. A point (x, y) can also be viewed as a complex number $c = x + yi$ (see Figure 3 (right)). From $x = \cos\theta$ and $y = \sin\theta$, complex number c is related to angle θ by exponentiation such that $c = \cos\theta + i\sin\theta = \exp(i\theta)$.

Parameterizing rotations. So far, we explained that complex numbers of unit length provide us with a non-singular parameterization of two-dimensional orientations. One might want to use complex numbers for parameterizing rotations as well for simplicity and uniformity. Unfortunately, complex numbers of unit length cannot parameterize rotations unambiguously. The ambiguity can easily be observed:

- A rotation can be partitioned into a series of partial rotations. The composition of partial rotations exhibits the trajectory of rotational movements. For example, a counter-clockwise rotation by angle π and a clockwise rotation by angle $-\pi$ travel different paths to end up with the same same destination, which is represented by complex number $\cos\pi + i\sin\pi = \cos(-\pi) + i\sin(-\pi) = -1$.
- A family of rotational actions by angle of $\theta \pm 2n\pi$ for any integer n represent a series of different rotational actions, but all of these actions are actually mapped to a single complex number $\exp(i\theta) = \exp\big(i(\theta \pm 2n\pi)\big)$ (see an example in Figure 3). This makes sense, since a rotation of $2n\pi$ about any fixed axis is equivalent to no rotation at all if we disregard the course of action and take account of only the result of action.

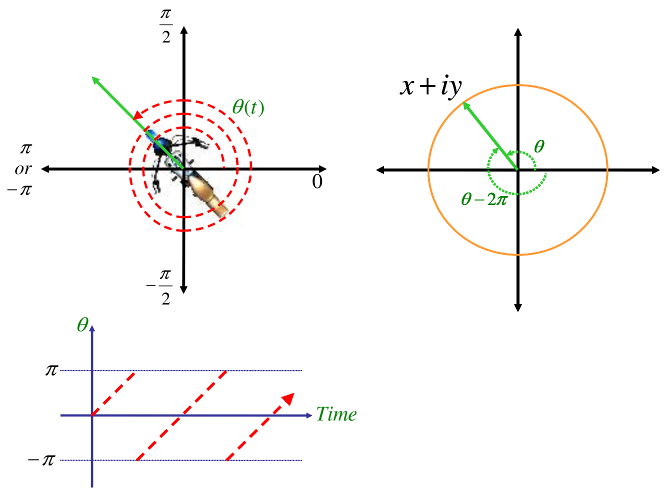

Figure 3: Rotations and orientations in two-dimensional space. The orientation of a two-dimensional rigid object can be represented by either a scalar angle from a reference orientation or a complex number of unit length. (Left and Bottom) By representing orientations by scalar angles, the orientation of a continuously-moving rigid object could be mapped to a time-varying function with discontinuous jumps at the boundary of the 2π-interval. (Right) The use of complex numbers can circumvent this singularity problem because any continuous motion can be mapped to a continuous complex function. However, complex numbers may not be appropriate for parameterizing rotations because many different angular motions (such as counter-clockwise rotation by angle θ and clockwise rotation by angle $\theta - 2\pi$) are mapped to the same complex number.

These examples show that parameterization by complex numbers is ambiguous and cannot distinguish rotations travelling different paths. This observation allows us to conclude that we have to use scalar angles for parameterizing rotations and complex numbers for orientations in two-dimensional space.

Is a rotation with an angle beyond 2π necessary for geometric computing? A rotation can be considered as a transformation that turns one orientation into another. All of such transforms can be represented by rotations with angles smaller than 2π. There can be two arguments about this question. The first argument is about the limited interval of rotation angles, which necessarily entails discontinuity issues. If the range of rotation angles is limited to any 2π interval, the parameterization includes discontinuous jumps at the boundary of the interval. Discontinuities are certainly undesirable for representing the circular nature of rotations. The second argument is about the interpolation of rotations. For example, a rotation by θ and another rotation by $\theta + 2\pi$ about the same axis produce different rotational actions when their trajectories are interpolated.

2.2 Three-dimensional space

The above arguments can be generalized to three-dimensional space by replacing complex numbers with unit quaternions and scalar angles with rotation vectors. As mentioned earlier, three-dimensional rotations can naturally be described by an axis and an angle, which form a rotation vector. There exists one-to-one correspondences between three dimensional rotations and vectors.

```
a.  (point) + (point) → (UNDEFINED)
b.  (vector) ± (vector) → (vector)
c.  (point) ± (vector) → (point)

d.  (point) - (point) → (vector)



g.  (scalar) · (vector) → (vector)

h.  (scalar) · (point) → (point)       if scalar=1
                       → (vector)      if scalar=0
                       → (UNDEFINED)   otherwise

j.  ∑ (scalar) · (vector) → (vector)
k.  ∑ (scalar) · (point) → (point)    if ∑ scalar=1
                         → (vector)   if ∑ scalar=0
                         → (UNDEFINED) otherwise
```

```
A. (orientation) · (orientation) → (UNDEFINED)
B. exp(rotation) · exp(rotation) → exp(rotation)
C. (orientation) · exp(rotation) → (orientation)
   exp(rotation) · (orientation) → (orientation)
D. (orientation)^-1 · (orientation) → exp(rotation)
   (orientation) · (orientation)^-1 → exp(rotation)
E. log(exp(rotation)) → (rotation)
F. log(orientation) → (UNDEFINED)
G. (scalar) · (rotation) → (rotation)
   exp(rotation)^(scalar) → exp(rotation)
H. (orientation)^(scalar) → (orientation) if scalar=1
                          → exp(rotation) if scalar=0
                          → (UNDEFINED)   otherwise
I. (rotation) ± (rotation) → (rotation)
J. ∑ (scalar) · (rotation) → (rotation)
K. affine_combi(orientations) → (ILL-DEFINED)
```

Figure 4: (Left) Operations between points and vectors in affine geometry; (Right) Operations between three-dimensional orientations and rotations. Orientations are represented by unit quaternions and rotations are represented by rotation vectors. Note that both `(orientation)` and `exp(rotation)` are unit quaternions, but represent different geometric entities.

Parameterizing orientations using rotation vectors suffers from either many-to-one correspondences or discontinuous jumps as we observed in the two-dimensional cases because the angle of rotation is limited within a 2π-interval for any given axis. This problem can be circumvented by using redundant parameterization. We can represent any three-dimensional orientation as a unit quaternion $\mathbf{q} = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} = (w,x,y,z) \in \mathbb{S}^3$. The redundancy is compensated by the unit length constraint $w^2 + x^2 + y^2 + z^2 = 1$.

Rotation vectors and unit quaternions can be converted to each other by using exponential and logarithmic mapping. Given a purely imaginary quaternion (a.k.a. rotation vector) $\mathbf{q} = (0, \mathbf{v}) = (0, \theta\hat{\mathbf{v}})$, quaternion exponentiation always produces a quaternion of unit length, which can be expressed in a closed-form: $\exp(0, \mathbf{v}) = (\cos\theta, \hat{\mathbf{v}}\sin\theta)$. The unit quaternion associated with a rotation about axis $\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$ by angle $\theta = \|\mathbf{v}\|$ is $\mathbf{q} = \exp(0, \mathbf{v}/2)$. We assume that unit quaternion $\mathbf{q}$ describes a rotation mapping $R_{\mathbf{q}}(\mathbf{u}) = \mathbf{q}\mathbf{u}\mathbf{q}^{-1}$ for $\mathbf{u} \in \mathbb{R}^3$. Here, vector $\mathbf{u} = (x,y,z)$ is interpreted as a purely imaginary quaternion $(0,x,y,z) \in \mathbb{S}^3$. Under this assumption, the left multiplication of a unit quaternion represents a rotation with respect to a fixed, world coordinate frame. Conversely, the right multiplication represents a rotation with respect to a local, moving coordinate frame. Two antipodal quaternions $\mathbf{q}$ and $-\mathbf{q}$ are mapped to a single orientation, but this mapping is still non-singular because the mapping is consistently two-to-one for any orientation.

Though the use of unit quaternions provides us with a useful non-singular parametrization for orientations, it cannot parameterize rotations without ambiguity. This can be shown as follows: Consider a family of rotations about axis $\hat{\mathbf{v}}$ by angle $\theta \pm 2n\pi$ for all integer n. All of these different rotational actions are mapped to a single unit quaternion or its antipode such that $\exp(0, \frac{\theta}{2}\hat{\mathbf{v}}) = \exp(0, \frac{\theta \pm 2n\pi}{2}\hat{\mathbf{v}})$ or $-\exp(0, \frac{\theta \pm 2n\pi}{2}\hat{\mathbf{v}})$. Note that both represent the same rotation. From these observations, we can conclude that we have to use vectors for parameterizing rotations and unit quaternions for orientations in three-dimensional space.

3 Operations between Rotations and Orientations

The operation between orientations and their relative rotations is coordinate-invariant if the result of the operation is independent of the choice of reference orientations. For example, consider two unit quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ representing two orientations of a rigid object in three-dimensional space. The multiplication of these two quaternions is computable, but the result depends on the choice of the reference orientation. This multiplication is neither coordinate-invariant nor corresponding to any tangible geometric meaning. It is analogous to the coordinate-dependence of point addition illustrated in Figure 1. We identified a set of coordinate-invariant operations between three-dimensional orientations and rotations (see Figure 4(right)). In this section, we will explain the operations one by one.

Unit quaternion $\mathbf{q} \in \mathbb{S}^3$ may represent either the orientation of a rigid object or its relative rotation. For clarity, we will use $\mathbf{q}$ only to represent the orientation, and the relative rotation will be denoted by the exponential of a rotation vector, that is, $\exp(0, \mathbf{v}/2) \in \mathbb{S}^3$. For notational convenience, we define two operators: $\mathbf{q} = \widetilde{\exp}(\mathbf{v}) = \exp(0, \mathbf{v}/2)$ and its inverse $\widetilde{\log}(\mathbf{q}) = \mathbf{v}$.

3.1 Multiplication

The multiplication of two unit quaternions indicates either the composition of two rotations or the rotation of a body from one orientation to the other. Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^3$ be vectors representing rotations. The composition of two rotations is computed as the multiplication of exponentials of two vectors (see Operation B in Figure 4): $\widetilde{\exp}(\mathbf{w}) = \widetilde{\exp}(\mathbf{u})\widetilde{\exp}(\mathbf{v})$. Note that $\mathbf{w} \neq \mathbf{u} + \mathbf{v}$, in general. It holds if vectors $\mathbf{u}$ and $\mathbf{v}$ are parallel.

Let $\mathbf{q} \in \mathbb{S}^3$ be a unit quaternion representing the orientation of a rigid body. Multiplication of $\widetilde{\exp}(\mathbf{v})$ and $\mathbf{q}$ indicates a rotation of the body about axis $\frac{\mathbf{v}}{\|\mathbf{v}\|}$ by angle $\|\mathbf{v}\|$ (see Figure 5). That brings the body to a new orientation $\mathbf{q}'$. Here, rotation vector $\mathbf{v}$ may be represented in a fixed coordinate frame such that $\mathbf{q}' = \widetilde{\exp}(\mathbf{v})\mathbf{q}$ or in a moving coordinate frame attached to the body such that $\mathbf{q}' = \mathbf{q}\widetilde{\exp}(\mathbf{v})$ (see Operation C in Figure 4).

3.2 Displacement

Given any two orientations $\mathbf{q}$ and $\mathbf{q}'$ of a rigid body, we can always find a single rotation that brings the body from one orientation to the other according to Euler's rotation theorem. The angular displacement $\mathbf{v}$ between given two orientations can be easily derived from $\mathbf{q}' = \widetilde{\exp}(\mathbf{v})\mathbf{q}$ or $\mathbf{q}' = \mathbf{q}\widetilde{\exp}(\mathbf{v})$ depending on in which coordinate frame we intend to represent $\mathbf{v}$: $\widetilde{\exp}(\mathbf{v}) = \mathbf{q}'\mathbf{q}^{-1}$ if $\mathbf{v}$ is represented in a fixed coordinate frame, and $\widetilde{\exp}(\mathbf{v}) = \mathbf{q}^{-1}\mathbf{q}'$ in a moving coordinate frame (see Operation D in Figure 4).

3.3 Exponential and Logarithm

The quaternion exponentiation is a many-to-one mapping and has a singular point at the antipole $(-1, 0, 0, 0) \in \mathbb{S}^3$. To define the inverse map, the domain is usually limited to $\|v\| < \pi$. Within the limited domain, the exponential map is one-to-one and thus its inverse map $\log : \mathbb{S}^3 \backslash (-1, 0, 0, 0) \rightarrow \mathbb{R}^3$ is well-defined.

From a geometry point of view, the logarithm of $\widetilde{\exp}(\mathbf{v})$ produces a vector describing a rotation (see Operation E in Figure 4). Note that, in general, $\mathbf{v} \neq \widetilde{\log}(\widetilde{\exp}(\mathbf{v}))$ because of the limited domain and range of the logarithmic map. For example, $\widetilde{\log}\left(\widetilde{\exp}\left((\theta + 2\pi)\hat{\mathbf{v}}\right)\right) = \theta\hat{\mathbf{v}}$ for any angle $\theta < \pi$ and unit vector $\hat{\mathbf{v}}$.

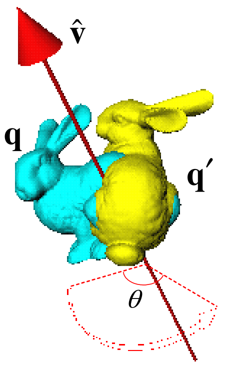

Figure 5: The relative rotation between any two orientations $\mathbf{q}$ and $\mathbf{q}'$ can be described as a vector $\mathbf{v} \in \mathbb{R}^3$ that specifies a rotation around a fixed axis $\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$ by the amount of angle $\theta = \|\mathbf{v}\|$.

Let $\mathbf{q} \in \mathbb{S}^3$ be the orientation of a rigid object. The logarithm of $\mathbf{q}$ can be interpreted as a rotation vector $\mathbf{w} = \widetilde{\log}(\mathbf{q})$. Since this vector is associated with a rotation from the reference orientation to the current orientation $\mathbf{q}$, the coordinates of $\mathbf{w}$ are necessarily dependent on the choice of the reference orientation (see Operation F in Figure 4).

3.4 Scalar Multiplication

Let $\mathbf{v} \in \mathbb{R}^3$ be a vector representing the spinning motion of a rigid object. Its scalar multiple $\alpha\mathbf{v} \in \mathbb{R}^3$ also represents a spinning motion around the same axis but the magnitude of its rotation angle is scaled by a factor of α (see Operation G in Figure 4). The power of $\widetilde{\exp}(\mathbf{v})$ to a scalar value α has the same geometric meaning since $\widetilde{\exp}(\alpha\mathbf{v}) = \widetilde{\exp}(\mathbf{v})^\alpha$. On the other hand, the power of orientation $\mathbf{q}$ to a scalar value α can be considered as $\mathbf{q}^\alpha = \widetilde{\exp}(\alpha\widetilde{\log}(\mathbf{q}))$, which is not well-defined because $\widetilde{\log}(\mathbf{q})$ is undefined (see Operation H in Figure 4). There are two exceptions where $\mathbf{q}^\alpha$ is well-defined. If $\alpha = 1$, $\mathbf{q}^1 = \mathbf{q}$ trivially. If $\alpha = 0$, $\mathbf{q}^0 = I$ where the identity quaternion $I = (1,0,0,0)$ indicates zero rotation.

3.5 Linear Combination of Rotations

The addition of two rotation vectors is yet another way of combining two rotations, which is different from applying two rotations one after the other: $\widetilde{\exp}(\mathbf{v})\widetilde{\exp}(\mathbf{u}) \neq \widetilde{\exp}(\mathbf{v}+\mathbf{u}) \neq \widetilde{\exp}(\mathbf{u})\widetilde{\exp}(\mathbf{v})$, in general (see Operation I in Figure 4). This operation has been used for decades without having a fundamental understanding of its geometric meaning, which was discovered recently by Alexa [1]. He showed that the addition of two rotation vectors allows the indicated rotations to be divided into infinitely small partial rotations and intertwined in such a way that $\widetilde{\exp}(\mathbf{u}+\mathbf{v}) = \lim_{n\to\infty}\left(\widetilde{\exp}(\mathbf{u})^{1/n}\widetilde{\exp}(\mathbf{v})^{1/n}\right)^n$. Since both scalar multiplication and addition of rotations are well-defined, their linear combination is also well-defined (see Operation J in Figure 4 and Figure 6).

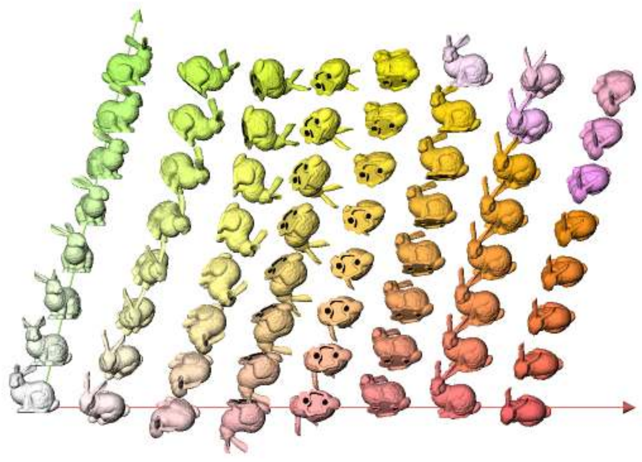

Figure 6: The linear combination of two rotations. The white bunny represents the reference orientation. The trajectories of rotations $\mathbf{v}_1 = (0, \pi, 0)$ and $\mathbf{v}_2 = (\frac{5}{2}\pi, 0, 0)$ are depicted along green and red axes, respectively. Linear combination $\mathbf{v} = t_1\mathbf{v}_1 + t_2\mathbf{v}_2$ are depicted in the range of $0 < t_1, t_2 < 1$.

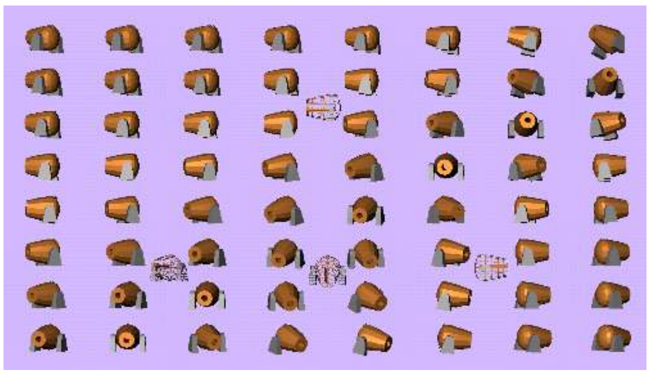

Figure 7: The affine combination of orientations. Each orientation on the grid is computed as an affine combination of four key orientations depicted as wireframes. Weight values are determined to provide a smooth parameterization on the xy plane using radial basis functions.

3.6 Affine Combination of Orientations

Loosely speaking, the affine combination is the weighted average of orientations, where the sum of weights equals one (see Figure 7). The affine combination of two orientations is well-defined and called *slerp* (spherical linear interpolation), which was coined by Shoemake [21]. The slerp can be implemented using three coordinate-invariant operations (Operations C, D, G in Figure 4) such that $\mathrm{slerp}_{t:1-t}(\mathbf{q}_1, \mathbf{q}_2) = \mathbf{q}_1(\mathbf{q}_1^{-1}\mathbf{q}_2)^t$. Therefore, the slerp is also coordinate-invariant.

Unfortunately, the affine combination of more than two orientations is ill-defined. In other words, it can be defined in several different ways and each definition produces different results. The definition by Buss and Fillmore [2] is widely accepted in the graphics community. The affine combination $\bar{\mathbf{q}}$ of orientations $\{\mathbf{q}_i\}$ with weights $\{w_i\}$ is defined by

$$\sum_i w_i \widetilde{\log}(\mathbf{q}_i \bar{\mathbf{q}}^{-1}) = (0,0,0),$$

where $\widetilde{\log}(\mathbf{q}_i \bar{\mathbf{q}}^{-1})$ is the displacement between $\mathbf{q}_i$ and $\bar{\mathbf{q}}$. According to this definition, $\bar{\mathbf{q}}$ can be determined uniquely if all $\mathbf{q}_i$'s lie in a hemisphere of $\mathbb{S}^3$. Though this definition is intuitive and rigorous, its practical use is limited because the evaluation of $\bar{\mathbf{q}}$ requires an iterative numerical solution. In the side box, we summarize briefly many other definitions that allow either closed-form or iterative solutions.

4 In Side Box

Since Shoemake [21] introduced quaternion Bézier curves to the computer graphics community two decades ago, many researchers have explored ways of constructing orientation curves, which are defined as the affine combination of key orientations. A number of methods have been developed and can be classified into several categories: Rectification, linearization, multi-linear, functional optimization, and incremental linearization. Among these methods, it is difficult to judge if one method is generally better than the others. No ultimate solution has been found so far and the adequate method may well be determined by the application.

Rectification. Since the unit quaternion space is not closed under addition and scalar multiplication, the weighted average of unit quaternions is, in general, not a quaternion of unit length. One popular approach is to compute the weighted average of unit quaternions as if they are four-dimensional vectors and then rectify the result to ensure the unit magnitude by simply normalizing its length. The affine combination of rotation matrices can be computed similarly by orthogonalizing the weighted sum of matrices [9]. These simple methods actually work reasonably well if all orientations are close to each other.

Linearization. The linearization method keeps quaternions on the unit sphere by using exponential and logarithmic maps [7, 8]. The basic idea is to transform orientation data into a vector space through logarithmic mapping, compute the affine combination of the transformed vectors, and then transform the result back to the unit quaternion space through exponentiation. A similar idea using rational maps has also been explored [10]. As mentioned in the previous section, the logarithm of a unit quaternion is coordinate-dependent if the unit quaternion represents an orientation. Hence, the linearization method is also coordinate-dependent and thus can be useful only if the application has an inherent coordinate frame which should not be changed.

Multi-linear. The affine combination of multiple points can always be reduced to a series of affine combinations between two points. From this observation, the multi-linear methods utilize a series of slerps for computing the affine combination of multiple orientations. A typical example is the construction of quaternion Bézier curves suggested by Shoemake [21]. The multi-linear method inherits the desirable coordinate-invariance property from the slerp. The downside of this approach is that the result is inconsistent because it is dependent on the order that the series of slerps is applied. The multi-linear method plays a useful role if a suitable ordering scheme is provided, such as the de Casteljau algorithm for Bézier curves [21] and the subdivision algorithm for B-splines [19].

Functional optimization. Several researchers addressed the problem from the viewpoint of functional optimization [2, 14, 20]. The affine combination of points can be viewed as the result of least squares minimization of a certain energy functional, which can be generalized for combining unit quaternions. The least squares minimization with unit quaternions does not allow a closed-form solution and thus uses an iterative numerical solver such as the Newton-Raphson method.

Incremental linearization. In many applications, unit quaternions to be combined are arranged in a sequence and indexed by a single integer index. The control points of orientation curves and the time-series of orientation samples are good examples. In those cases, a sequence of orientations can be described by a sequence of incremental rotations between each pair of successive orientations. The incremental linearization method uses the incremental rotations to avoid non-linear operations between orientations [11, 15, 18].

5 Extensions

5.1 Rigid body

Extending our formulation to rigid body motion is straightforward. The pose of a rigid object can be specified as its position and orientation. The spatial displacement of the object can be represented as a rigid transformation, that is, rotation followed by translation. Given rotation $\mathbf{q}$ and translation $\mathbf{v}$, the rigid transformation $T_{(\mathbf{v},\mathbf{q})}$ of any vector $\mathbf{u} \in \mathbb{R}^3$ is denoted by $T_{(\mathbf{v},\mathbf{q})}(\mathbf{u}) = \mathbf{q}\mathbf{u}\mathbf{q}^{-1} + \mathbf{v}$. The relation between the poses and displacements of a rigid object is analogous to the relation between orientations and rotations. We derive a set of operations for a rigid object shown in Figure 8 using six primitive operations between vector-vector and quaternion-vector tuples: multiplication, division, addition/subtraction, scalar multiplication, and the generalizations of exponential and logarithm maps. Letting $\mathbf{q} \in \mathbb{S}^3$ be a unit quaternion, $\mathbf{u},\mathbf{v} \in \mathbb{R}^3$ be three-dimensional vectors, and $\alpha \in \mathbb{R}$ be a scalar value, six primitive operations are

$$\begin{aligned}
(\mathbf{v}_1,\mathbf{q}_1) \otimes (\mathbf{v}_2,\mathbf{q}_2) &= (\mathbf{q}_1\mathbf{v}_2\mathbf{q}_1^{-1} + \mathbf{v}_1, \mathbf{q}_1\mathbf{q}_2) \\
(\mathbf{v}_1,\mathbf{q}_1) \oslash (\mathbf{v}_2,\mathbf{q}_2) &= (\mathbf{q}_2^{-1}(\mathbf{v}_1 - \mathbf{v}_2)\mathbf{q}_2^{-1}, \mathbf{q}_2^{-1}\mathbf{q}_1) \\
(\mathbf{v}_1,\mathbf{u}_1) \pm (\mathbf{v}_2,\mathbf{u}_2) &= (\mathbf{v}_1 \pm \mathbf{v}_2, \mathbf{u}_1 \pm \mathbf{u}_2) \\
\alpha \cdot (\mathbf{v},\mathbf{u}) &= (\alpha\mathbf{v}, \alpha\mathbf{u}) \\
\widetilde{\exp}(\mathbf{v},\mathbf{u}) &= (\mathbf{v}, \widetilde{\exp}(\mathbf{u})) \\
\widetilde{\log}(\mathbf{v},\mathbf{q}) &= (\mathbf{v}, \widetilde{\log}(\mathbf{q}))
\end{aligned}$$

Here, the multiplication is derived from the composition of two rigid transformations, which yields $T_{(\mathbf{v}_1,\mathbf{q}_1)} \circ T_{(\mathbf{v}_2,\mathbf{q}_2)}(\mathbf{u}) = \mathbf{q}_1(\mathbf{q}_2\mathbf{u}\mathbf{q}_2^{-1} + \mathbf{v}_2)\mathbf{q}_1^{-1} + \mathbf{v}_1 = T_{(\mathbf{q}_1\mathbf{v}_2\mathbf{q}_1^{-1}+\mathbf{v}_1,\mathbf{q}_1\mathbf{q}_2)}(\mathbf{u})$. The division is defined such that $\mathbf{p}' \oslash \mathbf{p} = \widetilde{\exp}(\mathbf{d})$ holds if $\mathbf{p} \otimes \widetilde{\exp}(\mathbf{d}) = \mathbf{p}'$, where $\mathbf{p},\mathbf{p}' \in \mathbb{S}^3 \times \mathbb{R}^3$ are the poses of a rigid object and $\mathbf{d} \in \mathbb{R}^3 \times \mathbb{R}^3$ is the displacement between these poses.

```
A. (pose) ⊗ (pose) → (UNDEFINED)
B. exp(disp) ⊗ exp(disp) → exp(disp)
C. (pose) ⊗ exp(disp) → (pose)
D. (pose) ⊘ (pose) → exp(disp)
E. log(exp(disp)) → (disp)
F. log(pose) → (UNDEFINED)
G. (scalar) · (disp) → (disp)
   exp(disp)^(scalar) → exp(disp)
H. (pose)^(scalar) → (pose)        if scalar=1
                   → exp(disp)     if scalar=0
                   → (UNDEFINED)   otherwise
I. (disp) ± (disp) → (disp)
J. ∑ (scalar) · (disp) → (disp)
K. affine_combination(poses) → (ILL-DEFINED)
```

Figure 8: Operations between the poses and displacements of a rigid object. The rigid pose (pose) is represented as a 2-tuple of a unit quaternion and a vector specifying the orientation and position, respectively, of the object. Its displacement (disp), measured with respect to a moving coordinate frame, is represented as a 2-tuple of vectors specifying the rotation and translation, respectively, of the object.

5.2 Articulated Figure

Our formulation can further be generalized to articulated figure motions. Let $\mathbf{P} = (\mathbf{v}_0, \mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n)$ be the pose of an articulated figure, where $\mathbf{v}_0 \in \mathbb{R}^3$ and $\mathbf{q}_1 \in \mathbb{S}^3$ are the position and orientation, respectively, of its root segment and $\mathbf{q}_k \in \mathbb{S}^3$ for $k > 1$ is the relative orientation of joint k with respective to its parent. The displacement between two articulated figure poses can be denoted by an array of angular and linear displacement vectors $\mathbf{D} = (\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_n) \in \mathbb{R}^{3(n+1)}$. We can define primitive operations between poses and displacements as follows:

$$\begin{aligned}
\mathbf{P}_1 \otimes \mathbf{P}_2 &= (\mathbf{q}_{1,1}\mathbf{v}_{2,0}\mathbf{q}_{1,1}^{-1} + \mathbf{v}_{1,0}, \mathbf{q}_{1,1}\mathbf{q}_{2,1}, \cdots, \mathbf{q}_{1,n}\mathbf{q}_{2,n}) \\
\mathbf{P}_1 \oslash \mathbf{P}_2 &= (\mathbf{q}_{2,1}^{-1}(\mathbf{v}_{1,0} - \mathbf{v}_{2,0})\mathbf{q}_{2,1}^{-1}, \mathbf{q}_{2,1}^{-1}\mathbf{q}_{1,1}, \cdots, \mathbf{q}_{2,n}^{-1}\mathbf{q}_{1,n}) \\
\mathbf{D}_1 \pm \mathbf{D}_2 &= (\mathbf{u}_{1,0} \pm \mathbf{u}_{2,0}, \cdots, \mathbf{u}_{1,n} \pm \mathbf{u}_{2,n}) \\
\alpha \cdot \mathbf{D} &= (\alpha\mathbf{u}_0, \cdots, \alpha\mathbf{u}_n) \\
\widetilde{\exp}(\mathbf{D}) &= (\mathbf{u}_0, \widetilde{\exp}(\mathbf{u}_1), \cdots, \widetilde{\exp}(\mathbf{u}_n)) \\
\widetilde{\log}(\mathbf{P}) &= (\mathbf{v}_0, \widetilde{\log}(\mathbf{q}_1), \cdots, \widetilde{\log}(\mathbf{q}_n)),
\end{aligned}$$

where $\mathbf{P}_i = (\mathbf{v}_{i,0}, \mathbf{q}_{i,1}, \cdots, \mathbf{q}_{i,n})$ and $\mathbf{D}_i = (\mathbf{u}_{i,0}, \mathbf{u}_{i,1}, \cdots, \mathbf{u}_{i,n})$. These basic operations combined with type-checking rules in Figure 8 allow us to identify coordinate-invariant operations for articulated figure motions.

6 Discussion

This work was partly motivated by our classroom teaching experiences. In computer graphics and animation courses, geometric programming with rotations and orientations seems one of the most difficult topics for lecturers to teach and for students to learn. The formalism presented in our work was developed as a means of explaining how to manipulate rotations and orientations in a geometrically meaningful manner. Although we have not yet attempted to assess the effectiveness of our formalism as a teaching aid, we believe that it has helped our students understand the basic theory and practical use of unit quaternions quite easily.

References

[1] Marc Alexa. Linear combination of transformations. In *Proceedings of SIGGRAPH 2002*, pages 380–387, 2002.

[2] Samuel R. Buss and Jay P. Fillmore. Spherical averages and applications to spherical splines and interpolation. *ACM Transactions on Graphics*, 20(2):95–126, 2001.

[3] Tony D. DeRose. Geometric programming: a coordinate-free approach. ACM SIGGRAPH 1988 Course #25 Notes, 1988.

[4] Tony D. DeRose. A coordinate-free approach to geometric programming (a geometric algebra and its implementation). Technical Report 89-09-16, Department of Computer Science and Engineering, University of Washington, 1989.

[5] Ron N. Goldman. Illicit expressions in vector algebra. *ACM Transactions on Graphics*, 4(3):223–243, July 1985.

[6] Ron N. Goldman. Vector geometry: A coordinate-free approach. ACM SIGGRAPH 1985 Course #16 Notes, 1985.

[7] F. Sebastian Grassia. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3(3):29–48, 1998.

[8] Gabriel Hanotaux and Bernard Peroche. Interactive control of interpolations for animation and modeling. In *Proceedings of Graphics Interface '93*, pages 33–42, 1993.

[9] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. Spatial keyframing for performance-driven animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 107–115, 2005.

[10] J. Johnstone and J. Williams. Rational control of orientation for animation. In *Proceedings of Graphics Interface '95*, pages 179–186, 1995.

[11] M. J. Kim, M. S. Kim, and S. Y. Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. *Computer Graphics (Proceedings of SIGGRAPH 95)*, 29:369–376, August 1995.

[12] L.Dorst and S.Mann. Geometric algebra: a computational framework for geometrical applications (part 1). *IEEE Computer Graphics and Applications*, 22(3):24–31, May/June 2002.

[13] L.Dorst and S.Mann. Geometric algebra: a computational framework for geometrical applications (part 2). *IEEE Computer Graphics and Applications*, 23(4):58–67, July/August 2003.

[14] J. Lee and S. Y. Shin. Motion fairing. In *Proceedings of Computer Animation '96*, pages 136–143, June 1996.

[15] Jehee Lee and Sung Yong Shin. General construction of time-domain filters for orientation data. *IEEE Transactions on Visualization and Computer Graphics*, 8(2):119–128, 2002.

[16] S. Mann, N. Litke, and Tony D. DeRose. A coordinate free geometry adt. Technical Report CS-97-15, Computer Science Department, University of Waterloo, 1997.

[17] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[18] F. C. Park and B. Ravani. Smooth invariant interpolation of rotations. *ACM Transactions on Graphics*, 16(3):277–295, 1997.

[19] D. Pletinckx. Quaternion calculus as a basic tool in computer graphics. *The Visual Computer*, 5:2–13, 1989.

[20] Ravi Ramamoorthi and Alan H. Barr. Fast construction of accurate quaternion splines. In *Proceedings of SIGGRAPH 1997*, pages 287–292, 1997.

[21] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of SIGGRAPH 1985*, pages 245–254, 1985.