Practical Parameterization of Rotations Using the Exponential Map

F. Sebastian Grassia
Carnegie Mellon University

The final version of this paper is published in *jgt, The Journal of Graphics Tools,* volume 3.3, 1998.
This reprint is included by permission of A K Peters, Ltd., publisher of *jgt*.

Abstract

Parameterizing three degree-of-freedom (DOF) rotations is difficult to do well. Many graphics applications demand that we be able to compute and differentiate positions and orientations of articulated figures with respect to their rotational (and other) parameters, as well as integrate differential equations, optimize functions of DOFs, and interpolate orientations. Widely used parameterizations such as Euler angles and quaternions are well suited to only a few of these operations.

The *exponential map* maps a vector in R^3 describing the axis and magnitude of a three DOF rotation to the corresponding rotation. Several graphics researchers have applied it with limited success to interpolation of orientations, but it has been virtually ignored with respect to the other operations mentioned above. In this paper we present formulae for computing, differentiating, and integrating three DOF rotations with the exponential map. We show that our formulation is numerically stable in the face of machine precision issues, and that for most applications all singularities in the map can be avoided through a simple technique of dynamic reparameterization. We demonstrate how to use the exponential map to solve both the "freely rotating body" problem, and the important ball-and-socket joint required to accurately model shoulder and hip joints in articulated figures. Examining several common graphics applications, we explain the benefits of our formulation of the exponential map over Euler angles and quaternions, including: robustness, small state vectors, lack of explicit constraints, good modeling capabilities, simplicity of solving ODE's, and good interpolation behavior.

1 Introduction

There are many ways to parameterize rotations. Why we would want to use a particular parameterization (or *any* parameterization at all) depends entirely on its performance in applications of interest. The primary applications of rotations in graphics are to encode orientations and describe and control the motion of rigid bodies and articulations in transformation hierarchies. Hierarchies, the backbone of most character animation systems, require not only free rotations, but also constrained one, two, and three degree-of-freedom (DOF) rotations whose angular range of motion is limited to more faithfully model the motion of, *e.g.* human joints (we'll explore ball-and-socket joints in section 5).

Parameterizing rotations for these applications is problematic mainly because rotations are non-Euclidean in nature (travelling infinitely far in any direction will bring you back to your starting point an infinite number of times). Any attempt to parameterize the entire set of three DOF rotations by an open subset of Euclidean space (as do Euler angles) will suffer from gimbal lock, the loss of rotational degrees of freedom, due to singularities[1] in the parameter space. Parameterizations that are themselves defined over non-Euclidean spaces (such as the set of unit quaternions embedded in R^4) may remain singularity-free, and thus avoid gimbal lock. Employing such parameterizations is complicated, however, since the numerical tools most often employed in graphics assume Euclidean parameterizations; therefore we must either develop new tools whose domains are non-Euclidean, or complicate our systems by imposing explicit constraints that

[1] Intuitively, a singularity is a continuous subspace of the parameter space, all of whose elements correspond to the same rotation – thus movement within the subspace produces no change in rotation.

distinguish the non-Euclidean parameter space from the Euclidean space in which it is embedded (as we must impose constraints that ensure quaternions retain unit length).

Every non-zero vector in R^3 has a direction and magnitude. We can associate a rotation with each vector by specifying the direction as an axis of rotation and the magnitude as the amount by which to rotate around the axis. If we augment this relationship by associating the zero vector with the identity rotation, the relationship is continuous, and is known as the exponential map. Unlike the quaternion parameterization, this parameterization is Euclidean, so it does contain singularities. The primary purpose of this paper is to show that for many common graphics applications, the singularities that cause gimbal lock in the exponential map are far away from the domain in which we must work, and that the resulting parameterization possesses most of the desirable qualities of the quaternion parameterization, without needing to worry about "falling off" the unit quaternion sphere (or any other non-Euclidean manifold). We will also discuss the strengths and weaknesses of the exponential map and more commonly used parameterizations as applied to several important graphics applications, to aid the practitioner in selecting the correct parameterization for the job.

In section 2 we examine how rotations are used in graphics applications and describe the pros and cons of the most commonly used parameterizations. Then in section 3 we develop the exponential map, and explain why it is advantageous to map into quaternions instead of mapping from R^3 directly to rotation matrices, and present formulae for computing quaternions and differentiating them with respect to R^3. In section 4 we show that this parameterization is extremely well suited to the applications of differential control and forward dynamics, and also discuss its limitations when applied to interpolation and spacetime optimization. In section 5 we further motivate the constrained three DOF rotation, and extend our method straightforwardly to handle it. We conclude in section 6 with a summary of the strengths and limitations of our formulation, and include in the appendices pseudocode and a pointer to supplemental C code for computing and differentiating using our formulation of the exponential map.

2 Evaluation of Common Parameterizations

There are five primitive means of describing and controlling rotations in graphics: forward kinematics (including keyframe interpolation), inverse kinematics, forward and inverse dynamics, and spacetime optimization. We will not consider other, higher level methods such as procedural animation and motion controllers, because they can generally be expressed in terms of the above primitives.

The applications built upon these primitives, several of which we will discuss in section 4, vary considerably in the size and complexity of the motion problems they address; however, they all require some or all of the following operations:

1. the ability to compute positions and orientations of points on body parts as functions of the parameters (*e.g.* position and direction of a humanoid's pointing finger), which is one aspect of forward kinematics
2. the existence of and the ability to compute derivatives of these positions and orientations with respect to the parameters, which is necessary for inverse kinematics, dynamics, and spacetime
3. the ability to integrate ordinary differential equations (ODEs) in parameter space, which is also required by inverse kinematics, dynamics, and spacetime
4. the ability to interpolate smoothly and controllably between sequences of parameter keyframes
5. the ability to combine rotations, either in the parameter space or in rotation space itself
6. the existence of an *inverse* operation that calculates parameter values from the corresponding rotation

Operations 3 and 4, and sometimes 5, naturally occur in the parameter space itself. Operations 1 and 2, however, are most conveniently carried out by expressing the rotation as a 4x4 transformation matrix, since this allows rotations to be included at any articulation in a transformation hierarchy right alongside translations, scales, and other linear transformations[2]. This gives us a common baseline for computation across various parameterizations: we will be interested in computing a rotation matrix and the partial derivatives of that rotation matrix with respect to its parameters. Therefore, if the parameterization possesses an n element vector of parameters $\mathbf{v}$, we must be able to compute:

[2] See Welman[11] for a method of computing Jacobians for hierarchies of only translations and rotations that does not involve transformation matrices. This is also an excellent introduction to inverse kinematics.

$$\mathbf{R}(\mathbf{v}) \text{ and } \frac{\partial \mathbf{R}}{\partial \mathbf{v}}$$

where $\mathbf{R}$ is a 4x4 matrix and $\partial\mathbf{R}/\partial\mathbf{v}$ is a 4x4xn tensor, that is, an n element vector of 4x4 matrices, each of which is the partial derivative of $\mathbf{R}$ with respect to one of the n parameters in $\mathbf{v}$. From these we can easily compute the jacobians of points and orientations with respect to the parameters, as demonstrated in the supplementary pseudocode (see Appendix).

Now that we know the quantities we are interested in computing, we can examine the parameterizations in use today and see where and why they are unsatisfactory.

2.1 3x3 Rotation Matrices

Each rotation can be represented as a 3x3 matrix whose columns are of unit length and are mutually orthogonal, and whose determinant is 1.0. The set of all such matrices forms the group SO(3) under the operation of matrix multiplication, and each member of the group in turn corresponds to a rotation. Since we have already stipulated that we are primarily interested in generating rotation matrices from our parameters, why not simply take the nine elements of the rotation matrix as our parameterization? If we were to do so, a rotation becomes a linear function of its parameters, which not only means that the rotation and its partial derivatives are trivial to compute, but also that we can potentially use linear optimization in our control algorithms, since positions and orientations on articulated figures will be linear functions of the parameters (translations, the other common transformation used in hierarchies, are also linear functions of their parameters).

Unfortunately, to optimize or differentially control using this parameterization, we must impose six non-linear constraints to ensure the matrix remains in SO(3) as its nine parameters are independently altered (three constraints to maintain unit length of all three columns, and three to keep them mutually orthogonal[3]). Similarly, each step taken while integrating an ODE will require that each rotation be re-orthonormalized.

2.2 Euler Angles

An Euler angle is a DOF that represents a rotation about one of the coordinate axes. There are three distinct functions $\mathbf{R}_x$, $\mathbf{R}_y$, and $\mathbf{R}_z$ for computing rotation matrices, depending on the coordinate axis about which the Euler angle rotates. These functions involve the sine and cosine of the Euler angle, and, although these functions are nonlinear, their derivatives are easy to compute.

The problems in applying Euler angles to our intended applications are well known in graphics [9]. Three DOF Euler rotations, formed by concatenating three single-axis rotations, suffer from gimbal lock when two of the three rotation axes align, causing a rotational DOF to be lost. This means there is a direction in which the mechanism whose orientation is being controlled by the Euler rotation cannot respond to applied forces and torques – it "locks up." It is straightforward to place limits on the legal range of motion for Euler angles, but since gimbal lock typically occurs when the second rotation in the chain has value 0 or $\pi/2$ (depending on the choice of Euler angles), we will not be able to avoid gimbal lock even for ball-and-socket joints, because, for example, shoulder joints require a range of rotation greater than $\pi/2$. Furthermore, interpolation of Euler angles results in poor interpolation of rotations, since Euler angles interpolate about each of the three axes independently, ignoring the interaction between the axes that arises from rotations' non-Euclidean nature. Euler angles are quite suitable for integrating ODEs, but since inverse kinematics, dynamics, and spacetime optimization require (at least) freedom from gimbal lock, Euler angles are unsuitable for these applications. We should note, however, that Euler angles provide an easy to use interface to animators in the form of three independent sliders (or the equivalent), and also work quite well in all applications requiring one or two DOF rotations.

---

[3] The seventh constraint defining SO(3), that determinant = 1.0 (as opposed to –1.0, the only other possibility), is subtly dependent on the other six, and generally does not require explicit enforcement.

2.3 Quaternions

Quaternions have a rich mathematics and history, including bitterly losing out to vector algebra as the accepted mathematical foundation for classical mechanics [8]. However, the reader of this paper will probably be familiar with quaternions, as presented to the graphics community by Shoemake [9], so we will only touch the highlights before discussing their strengths and weaknesses.

Quaternions form a group whose underlying set is the four dimensional vector space R^4, with a multiplication operator '$\circ$' that combines both the dot product and cross product of vectors [9]. The set of unit-length quaternions is a sub-group whose underlying set is named S^3. Quaternions are of interest to graphicists, roboticists, and mechanical engineers primarily because we can use S^3 to describe and carry out rotations. We do this by interpreting members of S^3 like so:

The quaternion $\mathbf{q} = [0,0,0,1]^T$ corresponds to the identity rotation.

The quaternion $\mathbf{q} = [q_x, q_y, q_z,\ q_w]^T$ encodes a rotation of $\theta = 2\cos^{-1}(q_w)$ radians about the unit axis $\hat{\mathbf{v}} = \frac{1}{\sin(\cos^{-1}(q_w))}[q_x, q_y, q_z]^T$.

In other words, we may parameterize a rotation of θ radians about the unit axis $\hat{\mathbf{v}} \in R^3$ with a unit quaternion constructed like so:

$$\mathbf{q} = [q_x, q_y, q_z,\ q_w]^T = [\sin(\tfrac{1}{2}\theta)\hat{\mathbf{v}},\ \cos(\tfrac{1}{2}\theta)]^T$$

The interesting thing is that if we want to rotate a vector $\mathbf{x} \in R^3$ by the rotation encoded in $\mathbf{q} \in S^3$, we can carry it out using only quaternion multiplication:

$$\mathbf{x}' = \text{Rotate}(\mathbf{x}) = \mathbf{q} \circ \tilde{\mathbf{x}} \circ \bar{\mathbf{q}}$$

where $\tilde{\mathbf{x}}$ is the vector $\mathbf{x}$ extended with a zero scalar component to make a quaternion, and $\bar{\mathbf{q}}$ is the conjugate of $\mathbf{q}$ ($\mathbf{q}$ with its vector part negated) [4]. One can prove that the result of the quaternion multiplications in this case will always have a zero scalar component, so the last step of the 'Rotate' function simply strips off the scalar part to arrive at $\mathbf{x}'$. Furthermore, there are simple formulae for computing a rotation matrix $\mathbf{R}(\mathbf{q})$ and its partial derivatives from a unit quaternion; one such is given by Shoemake [9][5]. The four partial derivatives $\partial\mathbf{R}/\partial q_x$, $\partial\mathbf{R}/\partial q_y$, $\partial\mathbf{R}/\partial q_z$, and $\partial\mathbf{R}/\partial q_w$ exist and are linearly independent over all of S^3, which means that unit quaternions are free from gimbal lock when used to control orientations.

However, this scheme relies on quaternions remaining in S^3 (*i.e.*, maintaining unit length) throughout the process of differential control, optimization, integration, etc. Since there are four directions in which a quaternion can change, but only three rotational DOFs, an optimizer or differential control algorithm is free to move the quaternion *off* the unit quaternion sphere, leading to non-rotations. Integrating ODEs in parameter space is also problematic because the instantaneous velocity or direction of change $\dot{\mathbf{q}}$ generally lies in the tangent plane to S^3, and any movement in the tangent plane to S^3 will push the quaternion out of S^3.

Several strategies have been developed to deal with these complications. The integration problem is generally addressed by re-normalizing the quaternion after every integration step, relying on small stepsizes to prevent the error from getting out of control. The "four derivative / three DOF" problem is typically dealt with in one of two ways. One can impose explicit constraints that force quaternions to maintain their unit length; this generally suffices (as long as these

---

[4] A slightly altered rotation formula uses the inverse of q instead of its conjugate, and produces rotations from non-unit quaternions as well. However, this formulation is undefined at the origin; furthermore, rotating vectors this way is not very convenient for use in transformation hierarchies, and results in far more complicated derivatives than using rotation matrices.

[5] The rotation matrix **M** presented in [9] transforms *row* vectors, so is actually the transpose of the matrix **R** that we desire for transforming column vectors. Our convention for rotating vectors by quaternions is also the opposite of that in [9], but is consistent with transforming column vectors.

constraints have higher precedence than other constraints that might be imposed), but increases the size of the systems we must solve, thus degrading performance. One can also use a function for $\mathbf{R}(\mathbf{q})$ that is defined over all of R^4 (except the zero element), like the *autonormalizing* formulae presented by Gleicher [2] (or, similarly, the function discussed in footnote 4). The problem with this method is that the Jacobian becomes rank deficient, which means that there is a direction in R^4 along which the quaternion can change without producing any change in orientation. One effect of this is to corrupt (make singular) the *mass matrix*, which is commonly used to achieve parameterization-independent scaling of simulated physical systems[12].

S^3 is an excellent place to interpolate rotations because it possesses the same local geometry and topology as $SO(3)$. Indeed, the *results* of interpolating with quaternions are generally pleasing to the eye, and can be made to possess desirable variational properties [9] [1] [6]. Recently, Kim *et. al.* [6] developed closed form quaternion curves on S^3 using Bezier, Hermite, B-spline (or indeed, any) blending functions, and were able to calculate high order parametric derivatives over the curves. This is great news for applications that must compute and optimize or integrate along fixed orientation curves, but it does not aid greatly in differential control or optimization over the curve shape itself, since it provides no correspondingly simple method for differentiating the curve with respect to the quaternion control points, and even if it did we would still face the inconveniences described in the preceding paragraphs. Nevertheless, the ability to specify closed form Hermite curves on S^3 by quaternion keys and angular velocities at the keys seems promising for use in key-frame animation systems, given suitable methods for visualizing the quaternion curves.

In summary, use of quaternions to parameterize rotations leads to numerically well-conditioned systems in the applications under consideration, but incurs an overhead in efficiency and/or code complexity whenever derivatives are used for control or optimization. Especially in light of recent developments, however, they may be the best choice for interpolation of three DOF rotations.

3 The Exponential Map

The problems we encounter with the quaternion parameterization arise because only a subspace of the full quaternion parameter space represents rotations. Given that we are interested in parameterizing a three DOF rotation, we would like a parameterization embedded in R^3 that is free of gimbal lock and interpolates rotations well using Euclidean interpolants such as cubic splines. This goal is, of course, unrealizable, as it is a standard exercise in topology to show that R^3 *cannot* be mapped into $SO(3)$ without singularities, *i.e.* gimbal lock. However, as we will now show, the inevitable singularities in the exponential map are often avoidable.

The exponential map maps a vector in R^3 describing the axis and magnitude of a three DOF rotation to the corresponding rotation. There are many different formulations of the exponential map[6]. One map popular in robotics texts is the *matrix exponential*, which maps R^3 into $SO(3)$ by summing an infinite series of exponentiated skew-symmetric matrices; the infinite series is generally evaluated using the compact *Rodrigues' Formula*[7]. However, we have found several advantages to using a map from R^3 to S^3, and using standard quaternion-to-matrix formulae for conversion to $SO(3)$. Among the advantages are that the inverse of the exponential map, the *log map* from S^3 to R^3, is much simpler than the log map from $SO(3)$ to R^3 (we require the log map to derive some important auxiliary quantities, for example see section 4.1), and that it may be useful to easily convert to and from S^3 when one needs to perform optimal interpolation (section 4.2) or spacetime optimization (section 0).

We can formulate an exponential map from R^3 to S^3 as follows:

$$e^{[0,0,0]^T} = [0,0,0,1]^T \qquad \text{and for } \mathbf{v} \neq \mathbf{0} \qquad e^{\mathbf{v}} = \sum_{m=0}^{\infty} (\tfrac{1}{2}\tilde{\mathbf{v}})^m = \left[\sin(\tfrac{1}{2}\theta)\hat{\mathbf{v}}, \quad \cos(\tfrac{1}{2}\theta)\right]^T ,$$

[6] Given that the idea of raising a scalar base to a non-scalar power is at best tenuously intuitive, why do we use the term exponential map? Historically, the formulation of the map as an infinite series has the same form as the series expansion of the exponential e^x for real numbers x, and thus the mathematical community has adopted the name *exponential* for the map.

where $\theta = |\mathbf{v}|$ and $\hat{\mathbf{v}} = \mathbf{v}/|\mathbf{v}|$, which maps $\mathbf{v}$ to a unit quaternion representing a rotation of θ (*i.e.* $|\mathbf{v}|$) about $\mathbf{v}$, where $(\frac{1}{2}\tilde{\mathbf{v}})^m$ is computed using quaternion multiplication. The formula on the right should look familiar: it is exactly the formula we used in section 2.3 to create a unit quaternion from a (unit) axis / angle description of a rotation. But now the exponential map has allowed us to encode both magnitude *and* axis of a rotation into a single three-vector.

The only problem with this particular formulation is that calculating $\hat{\mathbf{v}} = \mathbf{v}/|\mathbf{v}|$ as $|\mathbf{v}|$ goes to zero becomes numerically unstable. However, by rearranging the above formula a bit, we will be able to see that this exponential map *can* be computed robustly even in the neighborhood of the origin.

Let

$$\mathbf{q} = e^{v} = \left[\sin(\tfrac{1}{2}\theta)\frac{\mathbf{v}}{\theta},\ \cos(\tfrac{1}{2}\theta)\right]^{\mathrm{T}} = \left[\frac{\sin(\frac{1}{2}\theta)}{\theta}\mathbf{v},\ \cos(\tfrac{1}{2}\theta)\right]^{\mathrm{T}}$$

All we have done is reorganize the problematic term so that instead of computing $\mathbf{v}/|\mathbf{v}|$ (*i.e.* $\mathbf{v}/\theta$), we compute $\sin(\frac{1}{2}\theta)/\theta$. Why? Because $\sin(\frac{1}{2}\theta)/\theta = \frac{1}{2}\mathrm{sinc}(\frac{1}{2}\theta)$, and sinc is a function that is known to be computable and continuous at and around zero. Assured that the function *is* computable, we still need a formula for computing it, since sinc is not included in standard math libraries. Availing ourselves to the Taylor Expansion of sine, we get:

$$\begin{aligned}\frac{\sin(\frac{1}{2}\theta)}{\theta} &= \frac{1}{\theta}\mathrm{TaylorExpansion}(\sin(\tfrac{1}{2}\theta)) \\ &= \frac{1}{\theta}\left(\frac{\theta}{2} + \frac{\left(\frac{\theta}{2}\right)^3}{3!} - \frac{\left(\frac{\theta}{2}\right)^5}{5!} + \cdots\right) \\ &= \frac{1}{2} + \frac{\theta^2}{48} - \frac{\theta^4}{2^5 \cdot 5!} + \cdots\end{aligned}$$

From this we see that the term *is* well defined, and that evaluating the entire infinite series would give us the exact value. But as $\theta \to 0$, each successive term is smaller than the last, and terms are alternately added and subtracted, so if we approximate the true value by the first *n* terms, the error will be no greater than the magnitude of the *n*+1'st term. In fact, since machine precision is limited, we can evaluate the function with no *numerical* error like so:

When $\theta \le \sqrt[4]{\text{machine precision}}$ use just the first two terms of the expansion:

$$\frac{\sin(\frac{1}{2}\theta)}{\theta} = \frac{1}{2} + \frac{\theta^2}{48}\ ;$$

otherwise perform the actual sine computation and division by θ. Since all the dropped terms involve factors of θ, the approximation and actual function agree at $\theta = 0$.

3.1 Derivatives

Since we are in essence reparameterizing quaternions, we can compute the derivatives of $\mathbf{R}$ (the rotation matrix) with respect to its exponential map parameters by applying the chain rule. That is, we have $\mathbf{R}(\mathbf{q}(\mathbf{v}))$ and wish to compute $\partial\mathbf{R}/\partial\mathbf{v}$, which we can compute as the product:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{v}} = \frac{\partial \mathbf{R}}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial \mathbf{v}}$$

Since we already know how to compute the partial derivatives $\partial\mathbf{R}/\partial\mathbf{q}$, the only new quantities we need are the twelve partial derivatives of the quaternion with respect to its exponential map parameters (here $\partial\mathbf{q}/\partial\mathbf{v}$), which follow. Additionally, the appendix discusses the supplemental C source code for computing $\mathbf{R}$, $\partial\mathbf{R}/\partial\mathbf{v}$, and other quantities presented later in this paper.

To express the similarity in the form of the twelve derivatives, we let ℓ range over the three components of $\mathbf{q}$ that make up its vector part, and let *n* range over the components of $\mathbf{v}$.

The formulae for computing the partial derivatives of $\mathbf{q}$ with respect to $\mathbf{v}$ are, in the usual case where $\theta >> 0$:

$$\frac{\partial q_w}{\partial v_n} = -\tfrac{1}{2} v_n \frac{\sin(\tfrac{1}{2}\theta)}{\theta}$$

When $\ell = n$: $$\frac{\partial q_\ell}{\partial v_n} = \tfrac{1}{2} v_n^2 \frac{\cos(\tfrac{1}{2}\theta)}{\theta^2} - v_n^2 \frac{\sin(\tfrac{1}{2}\theta)}{\theta^3} + \frac{\sin(\tfrac{1}{2}\theta)}{\theta}$$

When $\ell \neq n$: $$\frac{\partial q_\ell}{\partial v_n} = \tfrac{1}{2} v_\ell v_n \frac{\cos(\tfrac{1}{2}\theta)}{\theta^2} - v_\ell v_n \frac{\sin(\tfrac{1}{2}\theta)}{\theta^3}$$

In the neighborhood of $\theta \to 0$, we can again replace sine and cosine by their Taylor expansions, and after simplifying, discard all terms with powers θ^4 or greater in the numerator. The result is the following forms for computing the partial derivatives of $\mathbf{q}$:

Let $\quad \mathrm{TSinc}(\theta) = \frac{1}{2} - \frac{\theta^2}{48} \quad \left(\text{The simplification of } \frac{1}{\theta}\mathrm{TaylorExp}\left(\sin\left(\tfrac{1}{2}\theta\right)\right) \right)$

$$\frac{\partial q_w}{\partial v_n} = -\tfrac{1}{2} v_n \, \mathrm{TSinc}(\theta)$$

When $\ell = n$: $$\frac{\partial q_\ell}{\partial v_n} = \frac{v_n^2}{24}\left(\frac{\theta^2}{40} - 1\right) + \mathrm{TSinc}(\theta)$$

When $\ell \neq n$: $$\frac{\partial q_\ell}{\partial v_n} = \frac{v_\ell v_n}{24}\left(\frac{\theta^2}{40} - 1\right)$$

3.2 Limitations

So far our formulation of the exponential map seems to fulfill all of our requirements, parameterizing an axis/angle rotation in three Euclidean parameters. Before we can discuss its application to the animation problems we have talked about, we must be clear about what we have given up (versus a straight quaternion parameterization).

3.2.1 Singularities

For the purposes of control and simulation, the principal advantage of quaternions over Euler angles is their freedom from gimbal lock. We already know that the exponential map must have singularities, so if it is to be useful, we must locate all singularities and show how they can be avoided at a cost that is outweighed by its benefits.

The exponential map has singularities on the spheres (in R^3) of radius $2n\pi$ (for n=1,2,3,…). This makes sense, since a rotation of 2π about *any* axis is equivalent to no rotation at all – the entire shell of points 2π distant from the origin (and 4π, *etc.*) collapses to the identity in SO(3). So if we can restrict our parameterization to the inside of the ball of radius 2π, we will avoid the singularity. Fortunately, each member of SO(3) (except the rotation of zero radians) has two possible representations within this ball: as a rotation of θ radians about $\mathbf{v}$, and as a rotation of $2\pi - \theta$ radians about $-\mathbf{v}$.

Because both control and simulation operate by moving through time in small steps, and the possible change in rotation on each step is small (certainly less than π), we can easily keep orientations inside the ball like so: at each time step when the rotation is queried for its value and derivative, we examine $|\mathbf{v}|$, and if it is close to π,[7] we replace $\mathbf{v}$ by $(1 - 2\pi/|\mathbf{v}|)\mathbf{v}$, which is an equivalent rotation, but with better derivatives. Such *dynamic reparameterization* could, in theory, also be applied to avoiding gimbal lock in Euler angles, but whereas here the reparameterization simply scales the current parameters, the corresponding operation on Euler angles involves switching the functions that define the rotation matrix and a sequence of inverse trig functions to determine the new parameters [10]. However, we must point out that this reparameterization deprives us of the ability to simply interpolate between successive state-snapshots produced by a simulator or inverse kinematics animation engine; before doing so we must make sure the consecutive rotation values are "close" to each other in R^3 (see section 4.2).

---

[7] We choose π rather than 2π because it gives us the maximal "buffer zone" against abnormally large steps, while still allowing all orientations to be representable.

3.2.2 Combining Rotations

One nice feature of quaternions is that the multiplication operator corresponds to matrix multiplication of rotation matrices. That is, if $\mathbf{q}_1$ and $\mathbf{q}_2$ are unit quaternions, then the combined rotation that is the result of first rotating by $\mathbf{q}_1$ and then by $\mathbf{q}_2$ corresponds to the unit quaternion $\mathbf{q}_3 = \mathbf{q}_2 \circ \mathbf{q}_1$. R^3 under the exponential map possesses no simple analogous operation. To compute $\mathbf{v}_3$, the vector corresponding to the combined rotation of first rotating by $\mathbf{v}_1$ and then by $\mathbf{v}_2$, we would need to map $\mathbf{v}_1$ and $\mathbf{v}_2$ to their corresponding quaternions, perform quaternion multiplication, then convert back, incurring several trig and one inverse trig functions.

Fortunately, this operation is typically not needed in the inner loops of the applications we have talked about. Rotations are changed only by direct parameter manipulation, or incrementally, via their derivatives. When they are combined, it is usually in the context of a transformation hierarchy, where all rotations have already been converted into transformation matrices.

4 Applications

Now that we have seen how the exponential map works and what its theoretical limitations are, it is time to focus on the reason we are presenting it: the simplification of algorithms that use parameterized rotations. Of the four applications we have discussed in this paper – control, simulation, optimization, and interpolation – we believe the exponential map to be very well suited to the tasks of differential control and simulation (forward dynamics plus integration). In the following sections we will explain why, and also discuss the complications that arise in applying it to interpolation and optimization.

4.1 Differential Control and Dynamics Simulation

One of the motivating applications for this work is differential control, which enables direct manipulation interfaces, inverse kinematics, and real-time control of robotic manipulators. To control the positions and velocities of objects and end-effectors of articulated assemblies, the only demands imposed by differential control are that the derivatives $\partial\mathbf{R}/\partial\mathbf{v}$ be continuous and free from gimbal lock. Since control is only performed at discrete instants in time, the simple dynamic reparameterization technique presented in section 3.2.1 will assure that these demands are always met.

In dynamics simulation applications, we track not only an object's instantaneous position and pose, but also its linear and angular velocity. Linear velocity is stored as a 3-vector that represents the Cartesian direction and magnitude of the velocity. Angular velocity is also represented as a 3-vector ω, whose meaning is nearly identical to that of the exponential map, except that its magnitude represents the *rate* of rotation about the axis rather than absolute orientation.

To update the position and orientation correctly as the simulation moves forward in time, we need, in addition to the derivatives necessary for differential control, a formula for mapping the instantaneous angular velocity into the tangent space of our parameterization. For a quaternion orientation the formula is the following [5]:

$$\dot{\mathbf{q}} = \tfrac{1}{2}\tilde{\omega} \circ \mathbf{q},$$

where $\tilde{\omega}$ is the angular velocity vector ω extended with a zero scalar component to make a quaternion. In the above formula and the subsequent ones, we have not explicitly denoted that $\mathbf{q}$ is actually a function of time, $\mathbf{q}(t)$, and thus $\dot{\mathbf{q}}$ is its time derivative.

To derive a similar formula for $\dot{\mathbf{v}}$, we make use of the inverse of the exponential map, dubbed, appropriately, the "log" map from S^3 to R^3, whose co-domain is rotations of magnitude less than π. Implicit differentiation gives us:

$$\mathbf{v} = \log(\mathbf{q}) = \frac{2\cos^{-1}(q_w)}{|\mathbf{q}_v|}\mathbf{q}_v, \qquad \text{Therefore} \qquad \dot{\mathbf{v}} = \frac{\partial \log(\mathbf{q})}{\partial \mathbf{q}}\dot{\mathbf{q}} \quad = \quad \frac{\partial \log(\mathbf{q})}{\partial \mathbf{q}}\tfrac{1}{2}\tilde{\omega} \circ \mathbf{q}$$

where $\mathbf{q}_v$ is the vector part of the quaternion. When we take the derivative of the log function with respect to $\mathbf{q}$ and put the entire formula on the right in terms of $\mathbf{v}$, much simplification occurs. We end up with the following formula for $\dot{\mathbf{v}}$ in terms of $\mathbf{v}$ and ω, which contains roughly the same number and kind of operations that the quaternion multiplication to compute $\dot{\mathbf{q}}$ would have required.

In the normal case where $\theta >> 0$:

Let $\mathbf{p} = \omega \times \mathbf{v}$ $\gamma = \theta \cot(\frac{1}{2}\theta)$ $\eta = \frac{\omega \cdot \mathbf{v}}{\theta}\left(\cot(\frac{1}{2}\theta) - \frac{2}{\theta}\right)$

Then $\dot{\mathbf{v}} = \frac{1}{2}(\gamma\,\omega + \mathbf{p} - \eta\mathbf{v})$

In the limit as $\theta \to 0$, Taylor expansions of $\cot(\frac{1}{2}\theta) = \cos(\frac{1}{2}\theta)/\sin(\frac{1}{2}\theta)$ simplify the above forms to:

Let $\mathbf{p}$ as above $\gamma_0 = \frac{12 - \theta^2}{6}$ $\eta_0 = \omega \cdot \mathbf{v}\frac{60 + \theta^2}{360}$

Then $\dot{\mathbf{v}} = \frac{1}{2}(\gamma_0\,\omega + \mathbf{p} - \eta_0\mathbf{v})$

We conclude that the exponential map is ideally suited to control and simulation because, despite the small measure we must take to avoid the singularity, it enjoys three advantages over the basic quaternion parameterization: 1) there is no need for explicit constraints when using jacobians of rotation-dependent quantities; 2) there is no need to re-normalize after integration steps; 3) our system's state vector will be smaller since each rotation requires three parameters vs. four.

4.2 Interpolation

Yahia and Gagalowicz [13] and later Hanotaux and Peroche [4] described methods for applying log maps to orientations to get them into R^3, where they then applied Euclidean cubic splines to interpolate between keys before applying an exponential map back into SO(3).

As discussed in section 2.3, interpolating in S^3 has several nice properties; among them is the fact that a geodesic interpolant (easily computed using the *slerp* presented by Shoemake [9]) between two orientations corresponds to the shortest path between the two orientations in SO(3). Hanotaux notes that the straight line between two orientations in exponentially mapped R^3 is not, in general, equivalent to the geodesic between the two orientations in S^3, but that "the approximation is not far from optimal." In fact the approximation can be quite far from optimal – quantifying how far is an open question, but in general the error increases the further the two axes of rotation diverge from parallel. Even if the axes of rotation of successive keys *are* close to parallel, the R^3 approximation will only approach optimality when one uses the proper log map, and since neither Yahia nor Hanotaux discusses this issue, we shall, briefly.

Each orientation in SO(3) maps to antipodes in S^3, *i.e.* a quaternion $\mathbf{q}$ and its negation, $-\mathbf{q}$. This means that to ensure that we get the geodesic interpolant out of a *slerp* between two orientations, we need only ensure that the two quaternions lie in the same hemisphere of S^3, *i.e.* their dot product is positive (if they do not we simply replace one of them with its antipode). However, a log map can map each orientation in SO(3) to an infinite number of points in R^3, corresponding to rotations of $2n\pi + \theta$ about axis $\mathbf{v}$ and $2n\pi - \theta$ about axis $-\mathbf{v}$ for any integer *n*. Given $r_1, r_2 \in \mathrm{SO}(3)$ and arbitrary log mapping $\mathbf{v}_1 = \log(r_1) \in \mathrm{R}^3$, in general only *one* of the infinitely many mappings of r_2 into R^3 will approximate the geodesic in S^3 when linearly interpolated from $\mathbf{v}_1$ in R^3. The procedure followed by Yahia [13] of limiting the range of the log map to $|\log(r)| \le \pi$ does not suffice. A log mapping that *does* guarantee the geodesic approximation picks the mapping for each successive key that minimizes the Euclidean distance to the mapping of the previous key.

Given such a log map that considers the previous mapping when calculating the current mapping, the results of interpolating in S^3 and R^3 may be visually indistinguishable for many applications, including keyframing. Furthermore (although we have no hard evidence or user tests to back this up), it may be more intuitive to control the shape of the interpolating curve using Euclidean Bezier and Hermite tangent knobs than by setting angular velocities at keys.

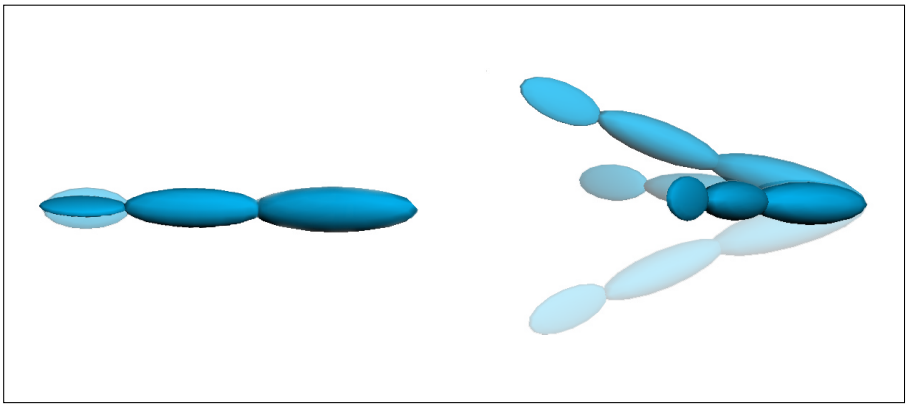

Figure 1: Degrees of Freedom for Ball-And-Socket Joint. The "arm" pictured above uses the ball-and-socket model proposed here at its shoulder joint. In the image on the left, the twist DOF is being exercised, and the arm spins about its axis. On the right the two swing DOFs are used to make the hand swing out a circle, starting at the bottom position and finishing closest to us. Note that in this entire motion there is no spin about the arm's axis.

4.3 Spacetime Optimization

In spacetime and other optimizations that operate over an entire animation simultaneously, DOFs are not simply angles at one instant in time, but rather rotation-valued functions of time. For instance, the function being optimized for a single joint-angle might be a cubic spline defined over the animation time interval, in which case the DOFs are the positions of the spline's control points, and we need to compute derivatives of orientations at various points in time (*i.e.* along the curve) with respect to these control points (simply several further applications of the chain rule to our existing formulae).

However, representing three DOF rotation functions in R^3 is fraught with peril because whenever the curve crosses one of the singularity shells discussed in section 3.2.1, some of the derivatives disappear. We cannot reparameterize the functions dynamically, because doing so will change the shape of the curves, potentially perturbing the optimization out of the state-space well it is currently traversing. Therefore if the possible range of rotations is greater than 2π (as for tumbling bodies), the exponential map is not well-suited as a parameterization. It should be noted, however, that many spacetime problems, including most humanoid character animation problems and any optimization that solves for motion *displacements* [3] rather than motions, operate only over the domain of rotations of magnitude less than 2π.

5 Ball-And-Socket Joints

Unconstrained three DOF orientations are of use primarily for rigid bodies whose entire state consists of a translation and a rotation. When we step up to articulated figures, however, the majority of the state is made up of internal joints, which are one, two, or three DOF rotations with *joint limits* that restrict the allowable angular motion. A somewhat crude but reasonable modeling of the joints in a humanoid breaks them down into *hinge*, *pivot*, and *ball-and-socket* joints. Hinge joints, such as the knee, would have one DOF and can be readily modeled using an Euler angle. A pivot joint, such as the wrist, would have two DOFs and can be modeled by two sequenced Euler angles.

Ball-and-socket joints, such as the joint between arm and shoulder, have three DOFs, which can be broken down into a twist about the limb axis, and a swing of the limb itself (see Figure 1). To model the motion accurately, it is important that we be able to limit the twist component independently of the swing[8]. While the limits on the twist component are generally small (within $\pm\pi/2$), thus making an Euler angle a reasonable choice for this DOF, the swing component (of human arms, at least) can reach well beyond $\pm\pi/2$. It is not possible to construct a single, three DOF Euler angle param-

[8] The twist and swing are not completely decoupled in human joints; however, the coupling we would get from using three Euler angles would, in addition to being prone to gimbal lock, be no closer to the true behavior than that of the decoupled model we present here. Furthermore, our model could easily accommodate controlled coupling based on kinesiological data.

eterization that does not experience gimbal lock at either 0 or $\pm\pi/2$ for the second angle in the chain, and, while quaternions will not lock up, it is difficult to place meaningful limits on their angular motion.

As already stated, the twist can be parameterized as either an Euler angle (if the principal axis for the limb happens to be a coordinate axis) or as a single DOF exponential map rotation, whose derivation follows straightforwardly from this presentation. The trick in parameterizing the swing component is to ensure that it has no rotational component about the twist axis. Fortunately, this is simple using an axis/angle parameterization like the exponential map, since a necessary and sufficient condition for a rotation that contains no spin about a specified vector is that the rotation axis be orthogonal to the vector.

This means that to achieve our desired two DOF swing rotation, all we need is an exponential map rotation vector that lies in the plane perpendicular to the major axis of the limb in its canonical (zero rotation) position. We do this by *reparameterizing* the rotation like so: pick any two orthogonal, unit vectors in the perpendicular plane; these vectors, $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ become a 2D basis for our desired swing rotation $\mathbf{v}$, an exponential map rotation that we compute like so:

$$\mathbf{v} = \alpha\hat{\mathbf{s}} + \beta\hat{\mathbf{t}}$$

where α and β are now the two DOFs, which we can gather into a 2D vector that we will call $\mathbf{r}$. Now since $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ are unit vectors, the length of $\mathbf{r}$ is the angular magnitude of the swing rotation, so to place a limit on the swing (*i.e.* the widest arc the limb can describe) we need only place an inequality constraint on this vector's magnitude:

$$|\mathbf{r}| \le \text{max allowable swing angle}$$

In fact, it is not much more difficult to impose a more flexible, ellipsoidal angular limit. If we know the angular limits of rotation along the major and minor axes of the ellipse, a and b, then we must choose $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ to correspond to the major and minor axes in the plane (for a limb that extends along the Z axis, $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ could be the X and Y axes), and pose the following inequality constraint instead of the one above:

$$\left(\frac{r_\alpha}{a}\right)^2 + \left(\frac{r_\beta}{b}\right)^2 \le 1$$

Since the two DOF rotation is formed from a reparameterization of the three DOF rotation, all the same formulae apply, but now we are computing the 3D vector $\mathbf{v}$ from the 2D vector $\mathbf{r}$. To compute the derivatives of $\mathbf{R}$ with respect to $\mathbf{r}$, we simply apply the chain rule once more:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{r}} = \frac{\partial \mathbf{R}}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial \mathbf{v}}\frac{\partial \mathbf{v}}{\partial \mathbf{r}} \qquad \text{where} \quad \frac{\partial \mathbf{v}}{\partial r_\alpha} = \mathbf{s} \quad \text{and} \quad \frac{\partial \mathbf{v}}{\partial r_\beta} = \mathbf{t}$$

This two DOF rotation is suitable for all the same applications as the three DOF, and additionally optimization. Recall that the three DOF rotation could not be used for optimization because dynamically reparameterizing the control points to avoid the singularity shells was unacceptable. But the first shell occurs at an angular magnitude of 2π, and since the angular motion limits for the types of joints the two DOF rotation models should always be much less than 2π, crossing any of the shells should never be a problem.

6 Conclusion

We believe that no single parameterization of rotations is best for all applications (in our own animation system we use the exponential map and Euler angles for inverse kinematics, and spherical Beziers with quaternions [9] for interpolation); this paper presents a robust method for computing the exponential map of three and two DOF rotations that outperforms other parameterizations for several important applications. We conclude with a summary of what we feel are its main strengths and weaknesses, and recommend it wholeheartedly to the implementor of inverse kinematics and dynamics simulation systems.

Strengths

- The exponential map remains free from gimbal lock over a range of axis/angle rotations up to magnitude 2π, which is suitable for any control or optimization algorithm that operates at single instants of time, provided time marches forward in small steps.

- The exponential map uses three parameters to parameterize SO(3), which means:
  - no need for normalization after integrating ODE's
  - no danger of falling out of a meaningful subspace (like falling off S^3 in R^4), so no need for explicit constraints
  - smaller state vectors, which combines with the previous point to result in faster performance
- The parameterization can be used to easily address the ball-and-socket joint problem, important in articulated figure animation.
- Interpolation using ordinary cubic splines is possible, and may often produce visually acceptable results provided successive keyframes are not too distant from each other in R^3.

Limitations

- The exponential map cannot be used for spacetime optimizations of tumbling bodies.
- There is no simple formula for combining rotations in R^3 akin to quaternion multiplication in S^3 or matrix multiplication in SO(3).
- The method used to avoid gimbal lock makes it impossible to simply interpolate between successive state snapshots output by a dynamics simulation or inverse kinematics engine (However, the output of such a system can be massaged into a form suitable for interpolation using the same technique we used for the log map in section 4.2.

Acknowledgements

Thanks to David Baraff for helpful discussions on singularities and computing $\dot{v}$, to Matt Mason and Mike Erdmann for providing a handle on the robotics knowledge base in this area, to Zoran Popović and Mike Gleicher for critical reads of this paper, and most especially to John Hughes for his Herculean efforts in giving the author a long-distance mathematical education sufficient to recognize and correct the many mathematical problems this paper originally contained.

Bibliography

1. Alan H. Barr and Bena Currin and Steven Gabriel and John F. Hughes. Smooth interpolation of orientations with velocity constraints using quaternions. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 331-320, July 1992.
2. Michael Gleicher and Andrew Witkin. Through the Lens Camera Control. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 331-340, July 1992.
3. Michael Gleicher. Retargeting Motion to New Characters. In Michael Cohen, editor, *Computer Graphics (SIGGRAPH '98 Proceedings)*, volume 32, pages 33-42, July 1998.
4. Gabriel Hanotaux and Bernard Peroche. Interactive Control of Interpolations for Animation and Modeling. In Tom Calvert, Program Chair, *Graphics Interface '93 Proceedings*, pages 201-208, May 1993.
5. David Hestenes. *New Foundations for Classical Mechanics*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1986, section 5-5 (equation 5.16).
6. Myoung-Jun Kim and Myung-Soo Kim and Sung Yong Shin. A General Construction Scheme for Unit Quaternion Curves with Simple High Order Derivatives. In Robert Cook, editor, *Computer Graphics (SIGGRAPH '95 Proceedings)*, volume 29, pages 369-376, August 1995.
7. Richard M. Murray and Zexiang Li and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, 1994, pages 22-34,73.
8. Richard P. Olenick and Tom M. Apostol and David L. Goodstein. *The Mechanical Universe : Introduction to Mechanics and Heat.* Cambridge University Press, New York, 1985, page 82.
9. Ken Shoemake. Animating Rotations with Quaternion Curves. In Brian A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 245-254, July 1985.
10. Ken Shoemake. Euler Angle Conversion. In Paul Heckbert, editor, *Graphics Gems IV*, Academic Press, 1994, pages 222-229.
11. Chris Welman. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation.* Master's Thesis, Simon Frasier University, 1993 (available from ftp://fas.sfu.ca/pub/cs/theses/1993/ChrisWelmanMSc.ps.gz).
12. Andrew Witkin and William Welch. Fast Animation Control of Non-Rigid Structures. *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 243-252, August 1990.
13. Hussein Yahia and André Gagalowicz. Interactive Animation of Object Orientations. Proceedings of the 2nd International Conference. Pixim 89. pages 265-75, September 1989; Paris, France.

Appendix: Sample Code

Sample C source code for computing the rotation matrix $\mathbf{R}$, its partial derivatives $\partial\mathbf{R}/\partial v$, and $\dot{v}(v,\omega)$ for the two and three DOF versions of exponential map rotations can temporarily be found at http://www.cs.cmu.edu/~spiff/exp-map. This site will eventually be moved to http://www.acm.org/jgt/papers/Grassia98. Also included at this site are supplementary documents containing a documented pseudocode function that uses the C code functions to compute the Jacobian contribution of a node in a transformation hierarchy with respect to all of the end effectors below it in the hierarchy (as one would in an inverse kinematics application).