Smooth Interpolation of Orientations with Angular Velocity Constraints using Quaternions

Alan H. Barr[†], Bena Currin[†], Steven Gabriel[††], John F. Hughes[†††]

California Institute of Technology[†]
Sage Design[††]
Brown University[†††]

Abstract

In this paper we present methods to smoothly interpolate orientations, given N rotational keyframes of an object along a trajectory. The methods allow the user to impose constraints on the rotational path, such as the angular velocity at the endpoints of the trajectory.

We convert the rotations to quaternions, and then spline in that non-Euclidean space. Analogous to the mathematical foundations of flat-space spline curves, we minimize the net "tangential acceleration" of the quaternion path. We replace the flat-space quantities with curved-space quantities, and numerically solve the resulting equation with finite difference and optimization methods.

1 Introduction

The problem of using spline curves to smoothly interpolate mathematical quantities in flat Euclidean spaces is a well-studied problem in computer graphics [BARTELS ET AL 87], [KOCHANEK&BARTELS 84]. Many quantities important to computer graphics, however, such as rotations, lie in non-Euclidean spaces. In 1985, a method to interpolate rotations using quaternion curves was presented to the computer graphics community [SHOEMAKE 85]; beyond this, there has been relatively little work in computer graphics to smoothly interpolate quantities in non-Euclidean, curved spaces [GABRIEL&KAJIYA 85]. In that paper, Kajiya and Gabriel developed a foundation for an "intrinsic" differential geometric formulation for computing spline paths on curved manifolds, and applied their results to quaternion paths.

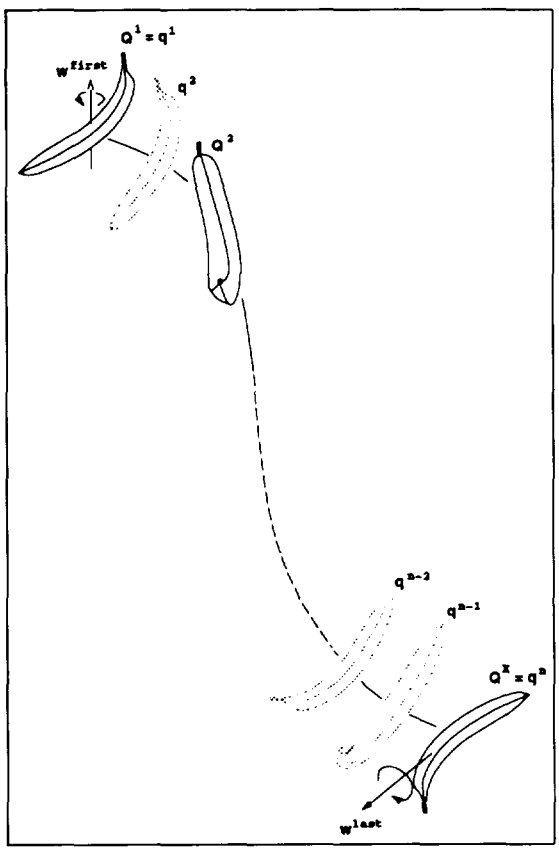

Figure 1. The interpolation problem we solve:

Given K keyframe quaternions, (capital) $\mathbf{Q}^i$, $i = 1, 2, \cdots, K$, at keyframe times $t_i = p_i\ h$, what are the n optimal interpolated quaternions $\mathbf{q}^{(p)}$, $p = 1, 2, \cdots, n$ at equally spaced times $\tau_p = h\ (p-1)$, that pass through the keyframe quaternions? $\mathbf{q}^{(p)} = \mathbf{Q}^i$ and $t_i = \tau_p$, when $p = p_i$. Optionally, find the n rotations (plus two extra keyframe rotations) when given angular velocities ω^{first} and ω^{last} of the first and last rotation along the path.

Splining in non-Euclidean Spaces

This paper presents a simpler version of the Gabriel/Kajiya approach to splining on arbitrary manifolds. Our approach uses extrinsic coordinates and constraints (rather than intrinsic methods, Christoffel symbols and coordinate patches), and generalizes to other manifolds that are embedded in Euclidean space.[1] The problem of computing spline curves on curved manifolds is of increasing importance to computer graphics, and we predict many future generalizations.

There are several reasons why someone would choose to use our interpolation techniques:

- The paths we generate through rotation space are very smooth.
- Our techniques allow the user to specify arbitrarily large initial and final angular velocities of a rotating body; by assigning large angular velocities, a user can make an object tumble several full turns between successive keypoints.
- It is fairly easy to add additional constraints.
- The techniques generalize to interpolations of other quantities in non-Euclidean spaces.
- The techniques are fast enough to experiment with, taking a few minutes per interpolation.

Of course, we cannot claim to have solved all problems of interpolating rotations and orientations. Through our choice of representation, we will have the classic advantages and disadvantages of using unit quaternions to represent rotations.[2] Also implicit in our approach is the assumption that the geometry of the space of orientations has a certain homogeneity, and that we can mathematically specify all of the constraints that we wish to apply.[3]

We find a path that minimizes a measure of net bending. We implement this, however, using a finite difference technique, so that we end up with a sequence of points on the path, rather than a continuous path. To produce a continuous path, we use Shoemake's *slerping* to interpolate between these points.

In section 2, we provide a brief discussion of quaternions, and present intuitive mathematical background to motivate the differences between interpolating in flat space and curved spaces; in section 3 we sketch the overall algorithm; in section 4 we present the constrained optimization problem; section 5 speaks briefly about numerical derivatives on manifolds; section 6 presents methods to solve the problem, while section 7 presents our results.

[1] Whitney's original embedding theorem tells us that every M dimensional manifold can be embedded in a $2M+1$ dimensional Euclidean space.

[2] The main advantage is that quaternion constraints are simple to enforce (constructing a four dimensional unit vector); the main disadvantage is *double representation*: there are two unit quaternions that represent each rotation.

[3] For tumbling bodies this is reasonable, but it is not completely true for camera orientations: certain orientations (ones with no "tilt" around line of sight of the camera) are far preferable to others. We would need to determine the appropriate constraints to minimize the net tilting.

2 Mathematical Background

Shoemake's paper on quaternions provides a good introduction to the mathematics of quaternions and their relationship to rotations. For our results, we need three basic facts about quaternions:

- The set of unit-length quaternions (i.e., expressions of the form $\mathbf{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ with $a^2 + b^2 + c^2 + d^2 = 1$) corresponds to the unit 3-sphere in 4-dimensions. The quaternion $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ corresponds to the point (a, b, c, d). The same quaternion is denoted by $\mathbf{q} = \begin{pmatrix} s \\ \mathbf{v} \end{pmatrix}$, where $s = a$ and $\mathbf{v} = (b, c, d)$.
- There is a natural map that takes a unit quaternion and produces a rotation: the quaternion $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ corresponds to a rotation of $2\cos^{-1}(a)$ about the axis (b, c, d) in 3-space. If $(b, c, d) = (0, 0, 0)$ the rotation angle is $2\cos^{-1}(\pm 1) = 0$, and the rotation is the identity.
- The map from unit quaternions to rotations is 2-to-1. For every rotation, two quaternions, $+\mathbf{q}$ and $-\mathbf{q}$, lying at antipodal ends of a hypersphere, correspond to it.

Advantages of quaternions. There are several reasons to use quaternions to describe rotations. First, the quaternion space has the same local topology and geometry as the set of rotations (this is *not* true of the space of Euler angles, for example, but is true of the 3×3 orthogonal matrices of determinant 1). Second, the number of coordinates used in describing a quaternion is small (4 numbers, in contrast to the 9 in a 3×3 matrix). Third, the number of constraints on these coordinates is small: the only constraint on a quaternion representing a rotation is that it have unit length; a 3×3 matrix must satisfy six equations to represent a rotation. Finally, the extrinsic equations for quaternions turn out to be fairly simple.

Disadvantages of quaternions. The main disadvantage of using quaternions is that their 2-to-1 nature necessitates a preprocessing step, to choose whether the plus or minus keyframe quaternion is the appropriate one to use.

Euclidean and non-Euclidean-space splines. Since the 3-sphere is a non-Euclidean space, we discuss interpolation methods for Euclidean spaces, and then motivate and describe a generalization to non-Euclidean

spaces. We will informally refer to them as "flat" spaces and "curved" spaces respectively.

2.1 Flat-space interpolation

The Hermite formulation expresses a spline curve as a parametric cubic curve $\gamma(t)$ that starts and ends at two given points[4], $\gamma(0) = \boldsymbol{P}^0$ and $\gamma(1) = \boldsymbol{P}^1$, and has given velocities there, i.e., $\gamma'(0) = \boldsymbol{R}^0$ and $\gamma'(1) = \boldsymbol{R}^1$. Given these boundary conditions (i.e., $\boldsymbol{P}^0$, $\boldsymbol{P}^1$, $\boldsymbol{R}^0$, and $\boldsymbol{R}^1$), we can find a unique cubic path that satisfies them. But why is a cubic the right curve to use?

One answer is given by reformulating the problem to ask "Among all curves starting at $\boldsymbol{P}^0$ with velocity $\boldsymbol{R}^0$ and ending at $\boldsymbol{P}^1$ with velocity $\boldsymbol{R}^1$, what curve bends the least?" We approximate the least square measure of curvature by minimizing the net squared length of the acceleration vector, γ''. Thus we seek to minimize

$$\mathcal{E} = \int_0^1 \gamma''(t) \cdot \gamma''(t) dt \qquad (1)$$

over all paths γ that satisfy the boundary conditions. The Euler-Lagrange equations [ZWILLINGER 89] provide a necessary condition for γ to be a minimum. Writing out these conditions gives $\gamma'''' = 0$, which means that each component of $\gamma(t)$ must be a cubic function of t.

A physical implementation of splines in a flat space. The word "spline" originally referred to a thin strip of wood or metal that was constrained by pins to form smooth curves for drafting or shipbuilding. For drafting, the pins were placed onto a flat surface; for shipbuilding, rigid posts were inserted into the earth, and wooden flexible planks were threaded between them. In each case, the splines flexed to meet the positional constraints imposed by the pins or posts. The spline took on curved shapes in its attempt to achieve a low-energy state, governed by equation (1).

2.2 Flat space splines versus curved-space splines.

We would like to carry out an analogous computation in a curved space: we define a "bending" measure of a curve, and then determine which curves minimize the measure. Unfortunately, the ordinary second derivative of a path is no longer the right way to measure net "bending." We can understand this by considering the problems that arise even for surfaces in 3-space.

If γ is a path on a surface M in 3-space, then γ can be thought of as a path in 3-space as well. As such, at each time t the path has a velocity vector $\gamma'(t)$ and an acceleration vector $\gamma''(t)$. Because γ lies within the surface, its velocity vector will always be tangent to the surface. Its acceleration vector, however, does not have to lie within the surface. It is likely to have components normal to the surface, as well as components tangential to the surface.

In Figure 2 we see a pair of curves on a surface. The midpoint of the upper curve has an acceleration vector **a** that points both out from the surface and up a little. We see that the acceleration vector **a** is not parallel to the surface normal N; the "non-N part" of vector **a** (the *tangential acceleration* or *covariant acceleration*) is labeled S in the drawing. The acceleration vector of the lower curve actually coincides with the normal vector to the surface, and hence its tangential acceleration is zero.

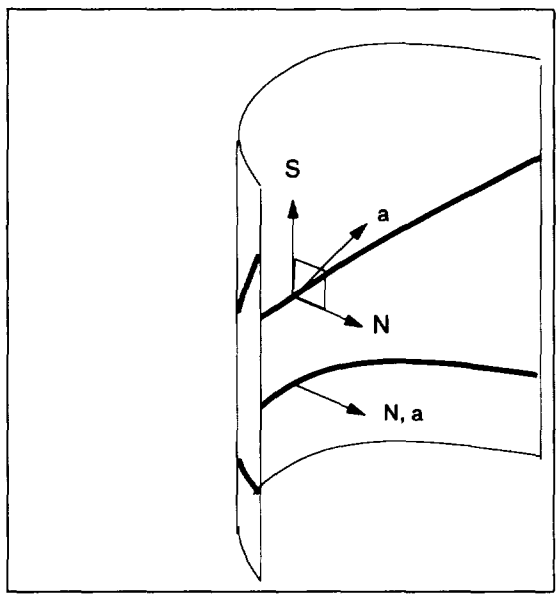

Figure 2. Two curves on a curved surface. The upper curve has an acceleration vector **a**, that does not lie in the surface. The vector **N** is the normal vector to the surface, along the path. The tangential part of the acceleration is the vector **S** = **a** \ **N** (described in section 2.3). The lower curve's acceleration is parallel to **N**, hence it has zero tangential acceleration.

A physical analogy. Imagine driving in a small circle in a hilly region. You feel two sorts of acceleration: you bounce up and down in your seat as you go over bumps, and you are pushed against your car door because you are turning in a tight circle. The first is acceleration in the direction normal to the surface of the earth; the second is the tangential acceleration. Note that if you want to take a drive, any path you take is likely to have some net tangential acceleration. But to make the trip as comfortable as possible, minimizing tangential acceleration is desirable.

Normal acceleration is inevitable. By contrast, the normal component of the acceleration is a necessary evil. Imagine trying to get from one place on a sphere to an-

[4] We use superscripts to indicate different vectors, and subscripts to denote x, y, z, etc components of vectors.

other in a way that minimizes total acceleration. If you travel along a great circle at a constant speed, the only acceleration will be normal. If you try to adjust your path so that you undergo *no* acceleration, you will have to be traveling in a straight line in 3-space, and hence will have to leave the surface of the sphere. This gets rid of the normal acceleration, but at the cost of violating the requirement that γ be a path *on* the surface.

Another physical example. Let us consider making a physical spline onto a spherical globe. Instead of placing pins into a flat drafting surface, we push the pins into the globe itself. We thread a semi-rigid elastic strip through the pins, making sure that the strip stays on the globe while being constrained by the pins. Since the strip needs to stay on the globe, we do not penalize it for bending to stay on the globe.

These examples motivate why we do not penalize acceleration normal to the surface, while penalizing acceleration within the surface, for constructing splines on curved surfaces. In generalizing Equation 1 to curved spaces, Kajiya and Gabriel therefore replaced the squared length of the acceleration vector with the squared length of the tangential acceleration. This is the starting point for our solution: we will seek a path in quaternion space, i.e., a path on the unit 3-sphere in 4-space, that minimizes the total squared tangential acceleration.

2.3 A formula for tangential acceleration

Given two n dimensional vectors $\mathbf{a}$ and $\mathbf{b}$, we wish to project away and remove all portions of $\mathbf{b}$ found in vector $\mathbf{a}$. The notation we use for this is $\mathbf{a} \setminus \mathbf{b}$ (read as vector $\mathbf{a}$ "without" vector $\mathbf{b}$). By definition,

$$\mathbf{a} \setminus \mathbf{b} = \mathbf{a} - \alpha\mathbf{b}, \text{ such that}$$
$$(\mathbf{a} \setminus \mathbf{b}) \cdot \mathbf{b} = 0$$
$$\text{which implies that } \alpha = \frac{(\mathbf{a} \cdot \mathbf{b})}{(\mathbf{b} \cdot \mathbf{b})}$$

If the surface M is a unit sphere, then the unit normal at the point (a, b, c) is (a, b, c). So for a path γ on the unit sphere, the total acceleration at time t is $\gamma''(t)$; its normal vector is $\gamma(t)$ itself, and the tangential acceleration $\mathbf{S}(t)$ is given by

$$\mathbf{S}(t) = \gamma''(t) \setminus \gamma(t).$$

For other applications, the formula for tangential acceleration of a curve on an arbitrary implicitly defined surface $f(x) = 0$ is

$$\mathbf{S}(t) = \gamma''(t) \setminus \mathbf{N}, \text{ where}$$
$$\mathbf{N} = \nabla f.$$

2.4 Physical meaning of paths on the quaternion sphere

We have already noted that each unit quaternion corresponds to a rotation. If we think of this rotation acting on a rigid body in a "home" coordinate system, then we can say that each quaternion corresponds to an orientation of the rigid body. Therefore a path in the quaternion sphere represents a continuously changing orientation. The derivative of the path at a particular instant represents the rate of change of orientation of the body, essentially its *angular velocity*. Thus to specify the endpoints and end tangents of a quaternion curve means to specify the initial and final orientations of a rigid body and its angular velocities at those points.

3 Algorithm Description

We provide a sketch of the overall algorithm in figure 3, using the curved-space results of the previous sections. In the subsequent few sections, we develop the mathematics for step 2. The implementation for step 2 is found in section 6.

1. Preprocess orientations into keyframe quaternions, $\mathbf{Q}^i$ as shown in Appendix A
2. Use constrained optimization techniques as described in section 6 to compute quaternions interpolated between the keyframes.
3. Optionally slerp between the interpolated quaternions to get a continuous representation.
4. Convert the quaternions back into rotation matrices (or other desired form).

Figure 3. The steps of the algorithm.

4 Mathematical Formulations

In this section, for our constrained optimization problem, we consider some of the merits of using a continuous derivative versus using discrete derivatives. Ultimately we will choose the discrete approach, because it is simpler. The reader should not infer that continuous approaches are not worthy of further investigation, however.

4.1 Continuous derivative approach

The problem statement for the continuous version without angular velocity constraints is: given K keyframe quaternions, $\mathbf{Q}^1$, $\mathbf{Q}^2$, $\cdots$, $\mathbf{Q}^K$, at times t_1, t_2, $\cdots$, t_K, what is the unit quaternion curve $\gamma(t)$ of minimal net least square tangential acceleration that passes through the points?

We are looking for the unknown (four dimensional) unit magnitude quaternion function $\gamma(t)$ which minimizes $\mathcal{E}$, the net square magnitude of the tangential acceleration. Without loss of generality,[5] we stipulate that $t_1 = 0$. Thus we minimize

$$\mathcal{E} = \int_0^{t_K} |\gamma''(t) \setminus \gamma(t)|^2 \, dt$$

subject to the constraints

$$\begin{aligned} \textbf{boundary values}: \quad \gamma(t_i) &= \mathbf{Q}^i, \quad i = 1, 2, \cdots, K. \\ \textbf{magnitudes}: \quad |\gamma(t)| &= 1, \quad 0 \le t \le t_K \end{aligned}$$

The boundary value constraints ensure that the quaternion path passes through the keyframe quaternions; the unit magnitude constraint keeps the quaternion on the unit 3-sphere. t_K and 0 are the (prescribed) values of t at the endpoints of the quaternion path.

This constrained optimization problem is a calculus of variations problem, which produces an Euler-Lagrange ordinary differential equation formulation with constraints [ZWILLINGER]. It is an extrinsic form of the Gabriel/Kajiya equation. The authors have derived this equation, but feel it would needlessly clutter the presentation. The approach involves the solution of a K-point ODE boundary value problem with constraints; we leave the pursuit of this approach as future work.

4.2 Discrete derivative approach

If we do not wish to solve K-point boundary value problems, we can make discrete approximations to convert the calculus of variations problem into a calculus problem. Instead of solving for an unknown function $\gamma(t)$, we solve for n fixed quaternions $\mathbf{q}^{(p)}$, $p = 1, 2, \cdots, n$. We retain the constraints that each $\mathbf{q}^{(p)}$ is a (four dimensional) unit vector, and that the appropriate $\mathbf{q}^{(p_i)}$s coincide with our keyframe quaternions $\mathbf{Q}^i$, $i = 1, 2, \cdots, K$.

We replace the continuous derivatives $\gamma(t)''$ in the $\mathcal{E}$ equation with a numerical approximation, shown in section 4.3; we denote the discrete derivative approximation with $(\mathbf{q}^{(p)})''$, and compute them from the $\mathbf{q}^{(p)}$s. In addition, we replace the integral with a discrete approximation, the sum of about n equally spaced values, times the stepsize, $h = t_K/(n-1)$.

[5] The reader can shift the arguments of the function to reduce a $t_1 \neq 0$ problem to a $t_1 = 0$ problem.

Thus, we minimize the function

$$E(\mathbf{q}) = h \sum_{p=p_{\min}}^{p_{\max}} \left| (\mathbf{q}^{(p)})'' \setminus \mathbf{q}^{(p)} \right|^2$$

subject to the constraints that

$$\begin{aligned} \textbf{boundary values}: \quad \mathbf{q}^{(p_i)} &= \mathbf{Q}^i, \quad i = 1, 2, \cdots, K \\ \textbf{magnitudes}: \quad |\mathbf{q}^{(p)}| &= 1, \quad p = 1, 2, \cdots, n. \end{aligned}$$

The p_i are those values of p where we wish the interpolated quaternions $\mathbf{q}^{(p)}$ to coincide with the keyframe quaternions $\mathbf{Q}^i$. $p_1 = 1$, and $p_K = n$; $p_{\min} = 1$ or 2 and $p_{\max} = n$ or $n-1$. They are chosen so that $(\mathbf{q}^{(p)})''$ can be computed in each term in the sum. (This is equivalent to having a weighting factor in the sum).

4.3 Discrete second derivatives

A simple discrete version of the second derivative is the *three-point* formula:

$$(\mathbf{q}^{(p)})'' = \frac{\mathbf{q}^{(p+1)} - 2\mathbf{q}^{(p)} + \mathbf{q}^{(p-1)}}{h^2}$$

We now have a calculus problem: find the n quaternions $\mathbf{q}^{(p)}$ that minimize the scalar function $E(\mathbf{q})$ subject to the above constraints. Without the angular velocity constraints we let $p_{\min} = 2$ and $p_{\max} = n - 1$.

4.4 Angular velocity constraints

Sometimes, we may wish to stipulate that angular velocities ω^{first} and ω^{last} apply to the first and last rotations along the path.

We can stipulate that the angular velocity is constant over the time interval $-h \le t \le 0$ and $t_K \le t \le t_K + h$. We reduce the problem with angular velocity constraints into the previous case, creating new quaternions and new constraints $\mathbf{q}^{(0)} = \mathbf{Q}^0$ and $\mathbf{q}^{(n+1)} = \mathbf{Q}^{K+1}$. To compute $\mathbf{Q}^0$, let

$$\begin{aligned} \omega &= \omega^{\text{first}}, \\ \theta &= h\,|\omega| \\ \hat{\omega} &= \omega/|\omega| \\ \mathbf{Q}^0 &= \begin{pmatrix} \cos(\theta/2) \\ -\sin(\theta/2)\,\hat{\omega} \end{pmatrix} \mathbf{Q}^1 \end{aligned}$$

To compute $\mathbf{Q}^{K+1}$, let

$$\begin{aligned} \omega &= \omega^{\text{last}}, \\ \theta &= h\,|\omega| \\ \hat{\omega} &= \omega/|\omega| \\ \mathbf{Q}^{K+1} &= \begin{pmatrix} \cos(\theta/2) \\ \sin(\theta/2)\,\hat{\omega} \end{pmatrix} \mathbf{Q}^K \end{aligned}$$

Thus, the method involving angular velocity constraints is merely a renumbered version of the previous

method. We let $p_{\min} = 1$ and $p_{\max} = n$, to add the two points. These points are the two smaller dots in figure 8.

5 Numerical derivatives on the 3-sphere

There are three problems that typically arise when using numerical methods to approximate derivatives on a manifold. First, some derivative formulas are not centered – they approximate the derivative, but not at the specified point. Secondly, there is a *numerical accuracy* problem – numerical approximations of the derivative typically will not lie in the tangent plane. Finally, there can be an *aliasing* problem, particularly for paths which circumnavigate the sphere or travel in tight loops. The aliasing problem greatly accentuates the numerical accuracy problem.

We compute our numerical derivatives using the centered three point formula for the second derivative shown in section 4.3. To solve the aliasing problem, we must choose n, the number of samples of $q^{(p)}$ to be large enough so that aliasing effects are not significant. To reduce aliasing, we suggest maintaining enough interpolation points so that adjacent $q^{(p)}$s do not travel more than $\pm 1/4$ way around the sphere, which can be tested via the condition $q^{(p)} \cdot q^{(p+1)} > 0$. For instance, between antipodal keyframe quaternions, two or more intervening interpolation points are needed.

For the angular velocity constraint, a similar condition suggests maintaining

$$|\theta| < \pi/2.$$

This implies that we need $n > \frac{2}{\pi}|d|\ t_{\max}$ steps, where $|d|$ is the magnitude of the larger of the two angular velocities.

6 Implementing the discrete derivative method

The most reliable way to implement the algorithm, whether or not angular velocity constraints are used, is to use a constrained optimization package for sparse systems, such as the MINOS package [MURTAGH&SAUNDERS 83]. Any method which solves for the $\mathbf{q}^{(p)}$ can be used, as long as it minimizes $E(\mathbf{q})$, subject to the constraints. By using first and second derivatives of the energy function $E(\mathbf{q})$, you can speed up the solutions significantly.

An advantage of this approach is that the packaged algorithms implement a robust convergence test, to determine when the optimal solution is found.

6.1 Augmented Lagrangian constraints

If the implementer does not wish to use prepackaged algorithms, a practical approach is to implement a variation of the Lagrangian methods in [PLATT 88], using first-derivative information. (We leave the implementation of faster methods, with quadratic convergence, as future work.)

First, you need the constraint function which keeps the p-th quaternion on the unit sphere

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

Then construct a total energy $F(\mathbf{q})$ by adding the constraint and penalty terms

$$F(\mathbf{q}) = E(\mathbf{q}) + \sum_{p=p_{\min}}^{p_{\max}} \lambda_p\ g_p(\mathbf{q}) + c\ (g_p(\mathbf{q}))^2$$

and take its derivative with respect to $q_\ell^{(r)}$ and with respect to λ_r

$$\frac{\partial}{\partial q_\ell^{(r)}} F(\mathbf{q}) = \frac{\partial}{\partial q_\ell^{(r)}} E(\mathbf{q}) + 2\sum_{p=1}^{n} (\lambda_p + c)\ q_\ell^{(r)}, \text{ where}$$

$$\frac{\partial}{\partial q_\ell^{(r)}} E(\mathbf{q}) = \frac{(q'' \setminus q)_\ell^{(r-1)}}{h^2} - \frac{2\,(q'' \setminus q)_\ell^{(r)}}{h^2} - \frac{q_j^{(r)}\, q_j^{(r)\prime\prime}\, (q'' \setminus q)_\ell^{(r)}}{q_k^{(r)2}} + \frac{(q'' \setminus q)_\ell^{(r+1)}}{h^2}$$

If $r \in [p_{\min} + 1, p_{\max} - 1]$, the above equation is valid. If $r = p_{\min} - 1$, only the $r+1$ term applies and the others are deleted; if $r = p_{\min}$, the $r+1$ and r terms apply, but the first term is deleted; if $r = p_{\max} + 1$, only the first term applies, while if $r = p_{\max}$, the first three terms apply.

Then, set up the differential equations

$$\begin{aligned} \frac{d}{ds} q_\ell^{(r)} &= -\frac{\partial}{\partial q_\ell^{(r)}} F(\mathbf{q}), \quad r \neq p_i \\ \frac{d}{ds} q_\ell^{(r)} &= 0, \quad r = p_i \\ \frac{d}{ds} \lambda_r &= +g_r(\mathbf{q}) \end{aligned}$$

and set up appropriate initial conditions:

$$\begin{aligned} q_\ell^{(r)}(0) &= Q_\ell^i \quad r = p_i, \quad i = 1, \cdots, K, \quad \ell = 0, 1, 2, 3. \\ q_\ell^{(r)}(0) &= \text{interpolated values between the } Q_\ell^i\text{s, either flat-space or results from previous runs with smaller numbers of points. Better initial conditions significantly improve the speed of this method} \\ \lambda_p &= 1 \end{aligned}$$

Numerically solve the differential equations with an automatic step-size method (such as Adams method), until you reach sufficiently constant values. This heuristic "stop" condition is why we advocate using packaged optimization algorithms, which have robust stop conditions.

It is recommended that the program be structured so that output from a smaller number of interpolated points can be used to set up the initial conditions for a run with a larger number of interpolated points.[6]

[6] You can also transform the variables in the differential equation via $s = 1 + 1/(\sigma - 1)$, essentially scaling the right hand side by $1/(1 - \sigma^2)$. The solution will then be found at $\sigma = 1$, rather than at $s = \infty$; you can iterate, numerically integrating the transformed differential equation repeatedly from 0 to 0.999 until the termination condition is reached.

7 Results

In the following figures, we show quaternion points visualized in three dimensions: we chose quaternions with the **k** component set to zero. Internally, of course, the implementation is fully four dimensional. We implemented augmented Lagrangian constraints, as well as a prepackaged version. The two methods agreed within the prescribed tolerances.

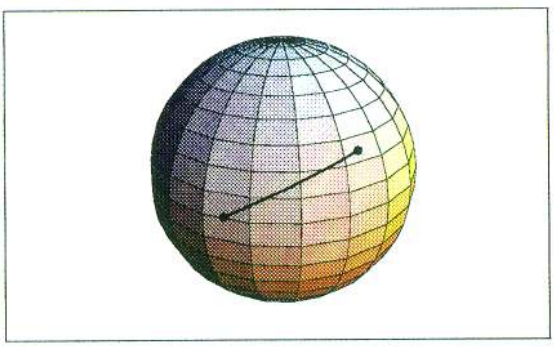

Figure 4a. Two keyframe rotations, without angular velocity constraints (shown as dots) on the interpolated path.

Figure 4b. The corresponding rotational path of the object. The two yellow objects are the two keyframe rotations, while the green images are the interpolated values. For clarity, we draw only a subset of the interpolated values.

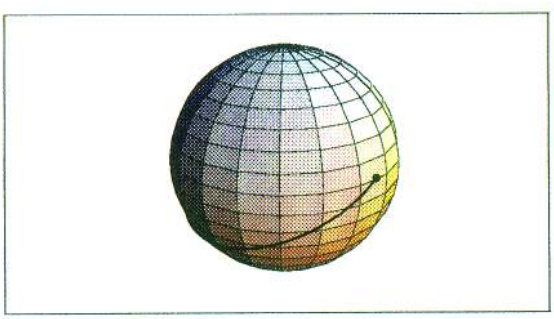

Figure 5a. We go half-way around the quaternion sphere for the same initial and final rotation, by choosing the antipodal point, $-\mathbf{Q}^1$. We rotate more fully around in space.

Figure 5b. The rotational path of the object in 5a.

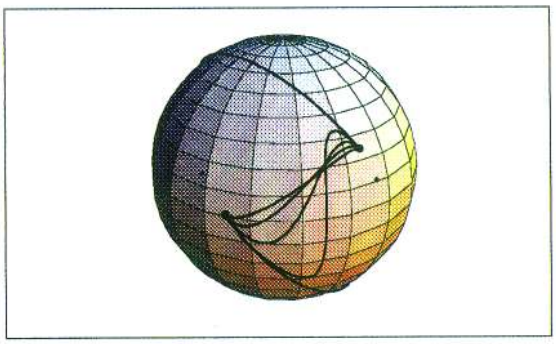

Figure 6a. Here we specify asymmetric angular velocity constraints, doubling until the path goes around the sphere.

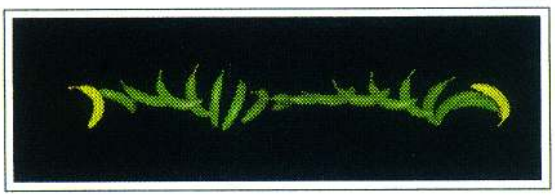

Figure 6b. The object rotates twice around.

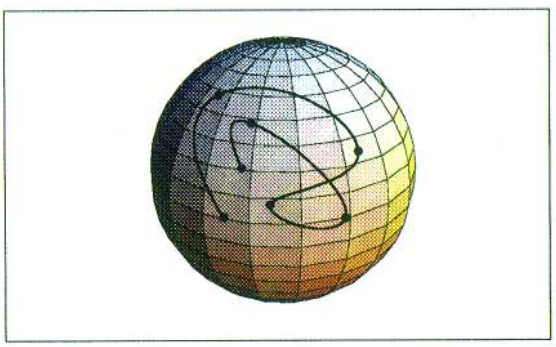

Figure 7a. Here we have seven keyframe quaternion points; there are 199 interpolated points.

Figure 7b. The seven keyframe rotations are clearly visible.

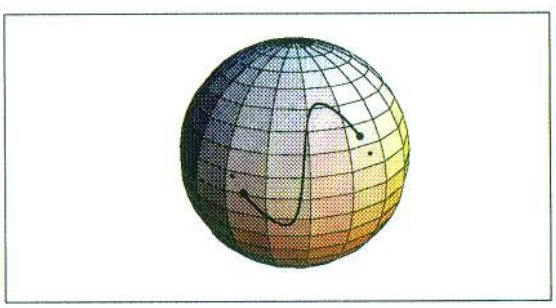

Figure 8. Symmetric angular velocity constraints are applied to the same endpoints in 4a. Note the two extra points, $\mathbf{Q}^0$ and $\mathbf{Q}^{K+1}$, drawn with smaller dots off of the curve.

Notes. In the figures, the keyframe quaternions are drawn with larger dots, while the keyframe quaternions from the angular velocity constraints are drawn with smaller dots. Note the qualitative similarity with flat-space splines. The banana rotates more in figure 5b than in 4b, due to the antipodal representation of the left rotation. Since the algorithm finds local minima, a different solution with a different number of loops might turn out to be the absolute minimum. The method, for large numbers of points, prefers good initial conditions, such as those produced by the algorithm with fewer points.

For figures 7a and 7b, 32 interpolation points are used in each interval, for a total of 199 points. The schedule of increasing points in each interval was $5 \Rightarrow 8 \Rightarrow 16 \Rightarrow 32$. The total computation time on an HP 700 was less than four minutes.

8 Conclusions

We have presented a new technique to smoothly interpolate rotations using quaternions. The method uses an extrinsic version of Kajiya and Gabriel's bend-minimization to characterize a spline in the quaternion 3-sphere; such splines are natural generalizations of splines in Euclidean space, and are particularly amenable to solution on the 3-sphere. We use a numerical method to determine several points between the key orientations; Shoemake's slerping can be applied to the points; the resulting splines are smooth, and have the desirable property that they pass through their control points exactly.

Our preliminary results are favorable, but there is much that can still be done to improve on this technique. We believe that splining in curved spaces will be of increasing importance to computer graphics, and predict many future generalizations.

Acknowledgements

The authors wish to thank Mark Montague, John Snyder, David Laidlaw, and Jeff Goldsmith at the Caltech graphics lab, as well as the Siggraph reviewers, for numerous helpful suggestions. The banana database is a generative model made by John Snyder and Jed Lengyel. This research was supported by the NSF/DARPA STC for Computer Graphics and Scientific Visualization, and by grants from HP, IBM, DEC and NCR to the university laboratories.

References

[1] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, Los Angeles, 1987.

[2] S. Gabriel and J. Kajiya. Spline interpolation in curved space. In "State of the Art Image Synthesis," Course notes for SIGGRAPH '85, 1985.

[3] W. R. Hamilton. *Lectures on Quaternions.* Hodges and Smith, Dublin, 1853.

[4] D. Kochanek and R. Bartels. Interpolating splines with local tension, continuity, and bias control. *Computer Graphics*, 18(3):33–41, July 1984.

[5] R. S. Millman and G. D. Parker. *Elements of Differential Geometry.* Prentice-Hall, Englewood Cliffs, NJ, 1977.

[6] B. A. Murtagh and M. A. Saunder. MINOS 5.0 user's guide. Technical Report SOL 83-20, Dept. of Operations Research, Stanford University, 1983.

[7] Ltd Numerical Algorithms Group. NAG Fortran library routine document, 1988.

[8] J. Platt. Constraint methods for flexible models. *Computer Graphics*, 22(4):279–288, July 1988.

[9] W.H. Press, B.P. Flannery, S.A. Teukolskym, and W.T. Vetterling. *Numerical Recipes in C...* Cambridge Univ. Press, Cambridge, England, 1988.

[10] K. Shoemake. Animating rotation with quaternion curves. *Computer Graphics*, 19(3):245–254, July 1985.

[11] M. Spivak. *A Comprehensive Introduction to Differential Geometry.* Publish or Perish, Inc., Boston, 1970.

[12] D. Zwillinger. *Handbook of Differential Equations.* Academic Press, San Diego, 1989.

Appendix A: Preprocessing Step to Create Spin

First, convert the K rotation matrices into K quaternions (see Shoemake or other quaternion reference for details). Then choose the desired spinning behavior of the objects between the quaternions. Sometimes, the object is desired to undergo an odd number of full spins around on an interval (usually once). These will be the "odd" intervals (and the other intervals are regarded as "even," which usually do not spin around). Multiplying a quaternion by -1 does not change the orientation it represents, but it does change whether or not an even or odd number of full-spins around the object takes place. The dot product of adjacent keyframe quaternions should be greater than or equal to zero for the even intervals, and less than zero for the odd ones. Multiply the quaternion by by -1 to change the interval from one state to the other.