Optimized Computer-Generated Motions for Animation

Thesis by

Jeff Goldsmith

In Partial Fulfillment of the Requirements
for the Degree of
Masters of Science

California Institute of Technology
Pasadena, California

1994

Abstract

Computer programmers working on computer animation have long been trying to solve the problem of how to move objects in user-desired ways with a minimum of user interaction. Objects moving from one place to another move along a path often determined by a spline. We would like to be able to allow the user a specify a characteristic of the object's motion and the animation system to choose a motion path that evidences that characteristic. We develop an approach using constrained optimization that will create paths. Some interesting motions have been found. We describe the effects obtainable from this method so that an animator can sensibly choose between them. We found that minimization of the covariant acceleration of all the points in a body leads to motion that is attractive. This motion seems to cause the moving body to anticipate its motion path in order to prevent sudden moves. It also seems to create very fluid-appearing motions because it tries to avoid sharp turns and sudden stops.

Acknowledgements

I would like to thank the entire Caltech Graphics Group for their help and for the environment in which this work was done. In particular, Bena Currin gave perceptive comments and did a wonderful critical reading of the manuscript. Al Barr was my advisor throughout the project and was instrumental in determining the direction of our search. Dave Kirk shared his experience in optimization. David Laidlaw gave me great advice that I wish I had followed more often. Ronen Barzel created the TeX macros and thesis environment that made putting this paper together easy. Mani Chandy gave me the moral support and confidence to finish this project. Without his generosity, I would not even have begun this project, and certainly not have finished it. Thanks, everybody.

Also thanks to the Society of Motion Picture and Television Engineers for their generous support during my time at Caltech.

This work was supported in part by grants from Apple, DEC, Hewlett Packard, and IBM. Additional support was provided by NSF (ASC-89-20219), as part of the NSF/DARPA STC for Computer Graphics and Scientific Visualization. All opinions, findings, conclusions, or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

Contents

Index of Figures

App. A. Notation

App. B. Derivations

App. C. Quaternion Arithmetic

Chapter 1

Introduction and Background

1.1 Introduction

Creating animation, whether using a computer or not, has always been a very time-consuming process. Not only does the creative talent need to make a myriad of decisions about what they want the final product to be, but the actual production of the many images is an enormous task. The advent of computer-aided animation has allowed computers to perform a great deal of the effort involved in this sort of work, but making computer-generated animation is still a laborious effort. We aim to find techniques that will help an animator achieve creative goals with less effort than before. The portion of the animation process that we address is motion design. Objects in computer animation need to have motion paths that meet animators' intuitive notions of what the objects will be doing and how they are going to be doing it. Perhaps most importantly, the techniques we are to create must be automatic; an animator must be able to make a menu selection and try out a type of motion. If he does not like it, he can try something different.

That goal is very general, so we have attempted a more modest accomplishment. We use a general technique for designing motion for a reasonably large set of scenes, though not for all scenes. The objects we have considered so far are articulated figures moving in an environment that does not exert any influence on them. Extending our approach to handle more complex environments is considered briefly in Chapter 3.

We also cannot hope to find tools to generate every sort of motion an animator could imagine; that is an impossible task. Instead, we investigated a set of motions and evaluated them with respect to two main guidelines. We asked, "is this motion smooth?" and "is this motion graceful?" We evaluate the answers to these questions empirically, but the motions themselves are computed numerically.

The approach we have taken is to use numerical constrained optimization to minimize some function over the space of all possible motions of articulated figures. We have constructed some functions that when so minimized, we feel create motions that are attractive, smooth, and graceful, albeit to varying degrees. The eventual product of this pursuit, we hope, is to create a menu of methods for animators to apply to their scenes. They can evaluate whether these motions are right for their work, and if not, try another one.

We build on the techniques of classical computer graphics. Most classical computer graphics animation was designed using key frame interpolation; our methods have the same inputs and user controls as does a classical key frame approach.

We also build on the work of other researchers who have considered this and related problems. We use as tools the techniques for representing rotations disseminated by Ken Shoemake [Shoemake] and have attempted to instantiate or generalize the work of Gabriel and Kajiya [Gabriel] and Barr *et al.* [Barr 92].

Throughout this work, we have represented equations in either the commonly-used vector notation or in Einstein Summation Notation (ESN). Almost all equations with subscripted symbols use ESN. A brief description of that notation is in Appendix A1. For further information, please see [Barr]. We have picked and chosen from the standard notation of vector mathematics and physics; most of the symbols we used are listed and defined briefly in Appendix A2.

1.2 Classical Computer Graphics Motion

1.2.1 Keyframe Animation and Splines

Before computers made their mark on animation, animation producers tried to solve the problem of needing to create huge amounts of original artwork in order to create animation. Animation is usually shown at 24 or 30 frames per second; each frame needs to be drawn and painted individually. Larger animation projects used a technique called *key frame animation.* The primary artists would produce artwork for the most important (key) frames of the animation, or every 30 frames or so. Lesser (cheaper) artists were hired to fill in the gaps by drawing the frames between the key frames. This made production of long movies practical.

Computers are perfect in the role of the lesser animator, but they are not clever enough to realize what motion was intended by the main animator, so they have to be told how to interpolate between the key frames.

Computer animation (in general) differs from conventional animation in that the computer is doing the drawing of the key frames, as well as of the intermediate frames. Usually, the computer is told the location and orientation of each of the objects in a frame that it is to draw. Since these are just numbers, the frames between the key frames can be drawn by choosing other numbers for the position and orientation of the objects and using the same program to draw the intermediate frames as was used to draw the key frames.

Finding numbers for intermediate frames is just a problem of interpolation. The position or orientation of an object can be considered to be a function of time. The drawing program is given values of this function at certain given times and needs to find values between the given ones.

The most commonly used technique for interpolating key frames is to put cubic polynomials between the key frames and just evaluate the cubic at any needed intermediate point. As a result, the overall function that represents the drawing parameters for a scene is often a piecewise cubic polynomial.

Many different cubics can be used to interpolate a set of points. In some cases, some properties are desirable for an interpolating function to have; in other cases, different properties are more important. Usually, no cubic is perfect, since most motions in the world are not cubic polynomials, but a good choice of interpolants will often get close to the desired motion.

1.2.2 Advantages

An automatically generated interpolant's two primary requirements are that it is easy to use and it is fast. The commonly used cubic polynomial schemes are definitely that. A user picks the points to be interpolated and pushes a button; a curve is drawn. Some curves are likely to be better than others, though. A few nice properties of an interpolant are: local control, the convex hull property, embedding of line segments, continuity of derivatives, exact interpolation of the given points, availability of additional shape parameters for user control and adjustment, and high speed of calculation.

Many of the current spline schemes have several of these properties. Local control means that changing one given point will only affect the cubic near the change, not others far from them.

The convex hull property loosely means that the interpolating curve will not diverge far from the given points. If the user wants part of the final function to be linear, he might like to embed line segments in the final interpolant; this is done automatically when colinear points are given to some splines. Most spline formulations enforce continuity of first derivatives, but not necessarily second derivatives. Some that produce second derivative continuity do not pass through the key frame values, but rather pass close by them. Beta-splines are an attempt at shape control; each interval is given two free parameters to adjust to get the curve closer to what a user had in mind when choosing the given points. Since these functions are all cubics, computing them is quite fast; a computer can often do it in real time. For far more detail about spline techniques, see [Bartels].

1.2.3 Shortcomings

Cubic interpolants have some drawbacks. Generally, not all of the nice properties mentioned above can be had at once; some have to be traded off in exchange for others. They also usually do not have second derivative continuity and almost never third derivative continuity; if that is a requirement for an application, cubic polynomial interpolation will not work. This drawback generalizes; if a user knows something about the interpolant he wants, a cubic spline is often not going to be able to supply that exactly. As a result, users often add huge numbers of key frames in order to get exactly what they want. We have even seen animations in which the user chose every frame to be a key frame, entirely defeating the interpolation mechanism. These interpolants also have no knowledge of what problem they are solving; they cannot be told about such influences as gravity or walls or other environmental conditions without express interference on the part of the user.

Perhaps the biggest drawback to cubic polynomial interpolation and basic splining techniques in general is that they are designed for interpolating within Euclidean space. If the space in which interpolation is done is not Euclidean, other techniques need to be found. See [Shoemake], [Barr 92], and [Gabriel] for some approaches to splining in non-Euclidean spaces. In particular, the space of orientations is non-Euclidean, so normal splining techniques need to be modified or replaced by ones derived for use in the space of orientations.

1.3 Newer Animation Techniques and Background

1.3.1 Physically-Based Modeling

Physically-based modeling is an approach to modeling that adds mechanical properties, such as mass, to geometric primitives and adds as primitives forces, torques, stresses, energies, and constraints in addition to the common geometric primitives. Objects' motions and other behaviors are determined from these primitives and their interactions. Physically-based modeling aims to simulate realistic motion of objects embedded in an active environment. Its primary advantages are that the behaviors so found are very difficult to produce using other techniques, and the ones it produces are often very realistic in appearance. Its primary disadvantages are that it is generally very difficult to use, and objects are often hard to control—if an object is desired to be somewhere at a given time, it is invariably late, early, or elsewhere. Perhaps this is another feature of the realism inherent in the approach. For more information on this subject, see [Barzel].

Our approach aims to combine some of the features of each of physically-based modeling and standard modeling. We try to control our objects with the same primitives as standard animators have, yet consider some of their mechanical properties in determining their motion.

1.3.2 Spline Interpolation in Curved Space

In the discussion above, the distinction between a spline and a cubic polynomial was left purposefully vague. In ordinary Euclidean space, they are identical; in other spaces, generally they are not.

Splines are curves that minimize the integral of the square of their covariant second derivative. In a sense, this measure is the "tension" of a curve, which is why cubic polynomials and not generally higher-degree polynomials are used for interpolation—the cubic gives the "best" interpolant, at least with respect to this one measure.

[Gabriel] gives the general form for curves that minimize tension in other spaces. Usually, this form simplifies only to a differential equation that must be solved for each interpolating function. They also consider the problem of splining in the space of rotations.

1.3.3 Quaternions

Until [Shoemake], in the world of computer graphics, nearly all rotations were described as a set of three successive rotations around the coordinate axes. Interpolation within that space is quite difficult; such problems as gimbal-lock and the presence of poles are common. Shoemake argued to represent rotations as quaternions, solving many of the problems encountered with axis-angle pairs.

Quaternions are an extension to complex numbers, using three square roots of -1. Normally, they are expressed as vectors with the real component first and the three imaginary components next.

The space of rotations maps well onto the space of quaternions. A rotation of θ around an axis v can be represented by the quaternion $(\cos(\theta/2), \sin(\theta/2)v)$.

We shall usually use quaternions to represent rotations. Some details of quaternion arithmetic are in Appendix C; more detail is available in [Shoemake] and [Shoemake 89].

1.3.4 Constrained Optimization

Numerical constrained optimization is a technique for finding a local minimum of a function within a space bounded by constraints. A constrained optimization problem consists of an objective function, a set (possibly the null set) of constraints, a space in which to search, and an initial point within that space. An optimizer looks for points within the search space with lower and lower values of the objective function, rejecting those that do not satisfy the given constraints. If points in all feasible directions (feasible by virtue of failing to violate any constraints) have higher values of the objective function, the optimizer stops. That point is a local minimum within the constrained search space.

Many algorithms exist for attempting to solve optimization problems. For a more thorough description of some of the simpler ones and a taxonomy of optimization problems, see section 2.4 or [Gill].

1.3.5 Spacetime Constraints

Spacetime Constraints is a technique for solving problems similar to the ones we are considering. Objects are constrained to be at given places at given times, and a motion path is constructed using optimization. In [Witkin], Witkin and Kass consider some motion path criteria and use as an example a complicated articulated body. In the last paragraph of that paper, the authors state that they had only begun to scratch the surface of ways to move bodies within a scene. We primarily consider the effect of different objective functions on the motion of bodies. In contrast, they emphasized the effect of the constraints on the motion of the objects in their scenes. Witkin and Kass minimized power of imagined muscular activity in their primary example. We consider some physical measures similar to power and some solely geometric measures for our optimizations.

1.3.6 Covariant Derivatives

Many of the objective functions we shall be attempting to minimize contain first and second derivatives of other functions. Since we are going to be computing these derivatives on manifolds embedded in larger spaces (due to constraints limiting our search spaces) we have chosen to consider the intrinsic geometry of the constraint manifold. The *covariant derivative* is the projection of the ordinary derivative into the tangent plane of the constraint manifold. A more precise definition follows.

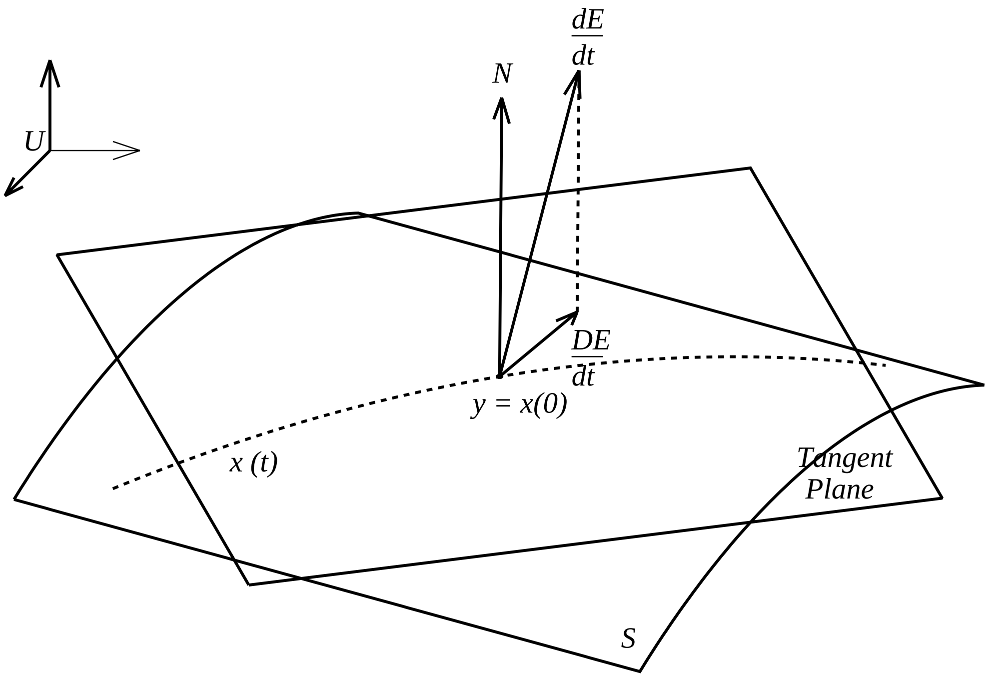

Figure 1.1: The Covariant Derivative of E at y (DE/dt.) □

Given a manifold S embedded in a larger space U, a vector field E defined upon that manifold, a curve $x(t)$ within the manifold and a point $y = x(0)$ on the curve, the vector obtained by projecting $dE(x)/dt$ at the point y into the tangent plane of S at y is called the *covariant* derivative of E at y. The covariant derivative is a feature of the vector field and the manifold only, and is not dependent on the curve, making it part of the intrinsic geometry of the field and the manifold. The covariant derivative is denoted DE/dt in contrast to the normal derivative dE/dt and can be computed from the unit normal to the manifold N and the ordinary derivative by

$$\frac{DE}{dt} = \frac{dE}{dt} - N(N \cdot \frac{dE}{dt}). \tag{1.1}$$

In our usage of the covariant derivative, S will be the search space limited by the constraints we give it. We shall use the covariant second derivative rather than the ordinary second derivative in many of our objective functions, because curves with accelerations parallel to the normal to the manifold (and thus with covariant acceleration zero) are geodesics on that manifold. While we shall not restrict our interpolants to geodesics, we shall consider evaluating closeness to a geodesic as beneficial for an interpolant. Minimization of the covariant second derivative will achieve that goal.

A geodesic is the shortest path between two points on a manifold. For example, the geodesics on a plane are straight lines. The shortest path is not often best for most purposes, but all other things being equal, a path closer to a geodesic would seem to avoid wasting effort compared to a path farther away from a geodesic. We shall try to penalize wasted motion; use of covariant derivatives will help measure wastage.

[Barr 92] considered the use of the covariant second derivative for optimization of rotation of a single body. For more information about covariant derivatives, see [Do Carmo] or [Misner].

Chapter 2

Optimized Computer-Generated Motion

2.1 The Scope of Problems We Address

Creating a technique for designing motion for any possible set of objects is an enormous task. Computer animation is used for animating objects as diverse as atoms, spacecraft, planets, dogs, and robots. It is possible that the general approach we consider here will eventually be useful for many of these types of animations, but we shall begin by restricting our attention to a very small subset of all possible objects.

2.1.1 Key Frame Animation

Animation can be created by any of a diverse set of techniques. Many are common to both traditional animation and computer animation. The most common throughout the computer-animation industry is called *key frame animation.*

Key frame animation is characterized by the presence of certain frames being created and established as unchangeable. These are the *key frames.* Frames between these are interpolated from the information of the key frames. Many techniques exist for performing this interpolation.

Our system will be given most of the information about the key frames before we start. An animator will supply the times of the key frames and the position and orientation of all the objects at the time of the key frames. In addition, the angular velocity of the objects can be chosen for key frames. Alternatively, it could be left to vary for some objects or for some key frames.

2.1.2 Interpolation of Key Frames

In traditional computer animation, splines are used to interpolate key frame animation. Our technique uses splines, also, but allows for a more general set of curves than a simple spline between key frames. Our general approach is to use optimization to create some in-between frames and to use splines to interpolate between them.

Many types of splines are known; each has advantages and disadvantages. In our case, we know (or have computed) the angular velocity of each object at the endpoints of each splined path. For the purpose, therefore, of interpolating orientation, we choose a Hermite polynomial to interpolate between computed points. As a result, our interpolation is a piece-wise cubic polynomial in each of the components of the quaternion representing the orientation of each object. This formulation will cause the interpolated values to represent unnormalized quaternions, but, since we shall not ever use quaternions for anything but rotation and shall avoid the undefined quaternion $(0,0,0,0)$, this will not be a problem. See Section 2.3.1 for more details about this. The number of cubic pieces of each quaternion component function is decided upon by the animator before the optimization step is begun. For the details of the Hermite polynomial interpolation, see Appendix B2.

2.1.3 Articulated Figures

The zoological definition of "articulated" is "having joints or composed of segments." Since our original intent was to find motions for objects that can be specified by adjectives, we chose to address a set of objects that vaguely includes animals or animal-like objects.

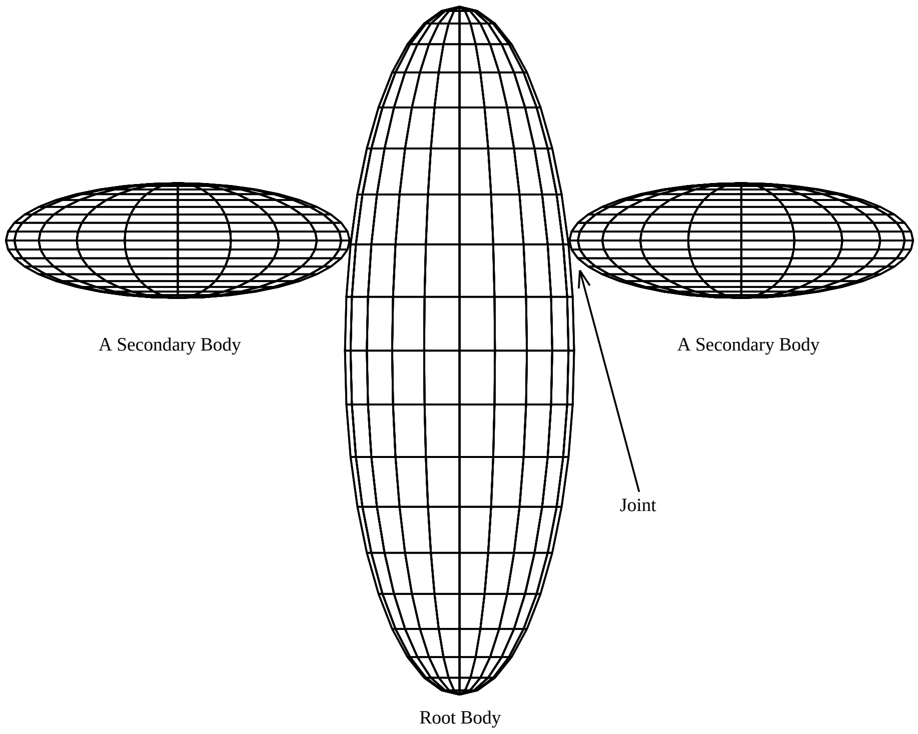

Figure 2.1: An Articulated Figure □

The restrictions we chose are that all objects are composed of simple geometric primitives, such as ellipsoids, cubes, and superquadrics. (See [Barr 81].) They must be connected at fixed points, around which the bodies are free to rotate in all directions, but they cannot separate. Each primitive is connected to exactly one "parent" body, with the exception of one body, connected to no parents, called the "root." Each body may have more than one child. As a result of this structure, only three degrees of translational freedom are left for the whole object. We just treat these as displacements of the center of mass of the root object. Since each body can rotate and the root can both rotate and translate, the total number of degrees of freedom of one of these articulated figures is $3(n+1)$ where n is the total number of bodies in the object.

2.2 Functions to Minimize

We have considered many different energy functions to minimize. They were chosen with the intent of obtaining smooth graceful motion or something interesting and close to that goal.

2.2.1 Single-Body Objective Functions

The set of single bodies is a subset of the set of articulated bodies. Since our methods apply to the larger set, they are also applicable to single bodies. Most prior work centered on finding schemes to interpolate characteristics of single bodies. Shoemake [Shoemake] was probably the first to consider how to interpolate rotations of a single body. He used a spline in the space of quaternions to do this. Gabriel and Kajiya [Gabriel] extended this idea to the general case, deriving the general form of splines for most spaces. The space of degrees of freedom of articulated bodies meets their criteria for spaces in which to compute splines, but we chose to consider energy functions other than the one necessary for cubic splines. In 1992, Barr *et al.* [Barr 92] considered the problem of using optimization to interpolate rotations of a single body. Their method produced smooth rotations with some interesting characteristics, not the least of which is that they were able to make bodies rotate multiple times around between two key frames just by specifying angular velocity at the key frames.

We have attempted to expand upon this idea by choosing different energy functions to minimize.

Kinetic Energy

The first energy function that came to mind as perhaps creating attractive motion is the kinetic energy of the body. Minimizing kinetic energy would seem to produce a very efficient smooth motion—one that wastes as little movement as possible.

For a single rotating body, the kinetic energy is

$$KE(t) = \frac{1}{2}\omega_i(t)M_{ik}(t)\mathcal{I}_{kl}M_{lj}(t)\omega_j(t), \tag{2.1}$$

where $\mathcal{I}$ is the moment of inertia tensor for the body, M_{ij} is the orientation of the body expressed as a rotation matrix, and ω_i is the angular velocity of the body. We minimized

$$E = \int_{t_{min}}^{t_{max}} KE(t)dt. \tag{2.2}$$

Since the moment of inertia of a rigid body in its home coordinates is constant over time, minimization of this quantity is similar to a least-squares minimization of angular velocity. The least squares fit of angular velocity to unconstrained points is a straight line, so the minimum occurs when angular velocity is as close to constant as possible subject to the body's reaching the orientations at the key frames.

Constant angular velocity, while a simple solution, is generally not very interesting motion. Even if it were, numerical optimization is hardly a computer-time efficient method to calculate it.

Kinetic energy, however, seems like an important physical quantity for rotating objects. While we have concluded that we do not want to minimize it, kinetic energy might be useful combined with other objective functions. Perhaps a constraint that a body's kinetic energy stays below some fixed amount may be useful, for example.

Figure 2.2 shows a motion path generated by minimization of kinetic energy. The thick black curve represents the function $q(t)$, the quaternion representing the orientation of a single body (an ellipsoid similar to those shown in the earlier figures.) The space of quaternions is a four-dimensional hypersphere, so in order to display the quaternions in three dimensions, one component was projected away; the actual results of the optimization were such that the ignored values were all very close to zero. The white dots are the three key frames that the program interpolated. Other experiments will be displayed similarly; in each case, the component of the quaternions that is closest to zero is projected away. In each case, key frame values were chosen so that one component would stay close to zero. The angular velocities at the two endpoints in this example were constrained to be zero.

Note the sudden change in direction of $q(t)$ in Figure 2.2 at the center key frame.

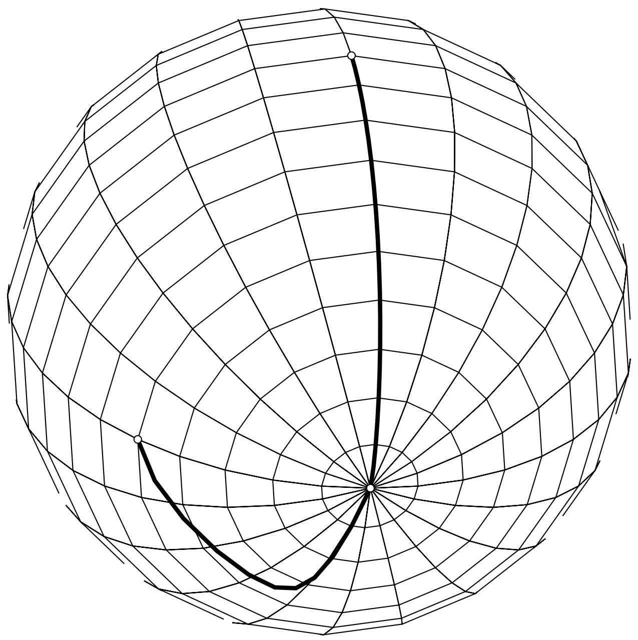

Figure 2.2: Motion path created by minimizing kinetic energy; the white dots are key frames. Shown in a projection of quaternion space. □

Acceleration

The square of the angular velocity is not a very interesting quantity to minimize, but the square of the derivative of the angular velocity, the angular acceleration, is much better. We next tried to minimize the square of the length of the quaternion acceleration.

Representing the orientation of the body as a quaternion, $q(t)$, we minimized

$$E = \int_{t_{min}}^{t_{max}} \ddot{q}_i \ddot{q}_i \, dt \tag{2.3}$$

subject to the soft constraint that the length of the quaternion vector was unity. This leads to very smooth motions because any sharp changes in motion lead to large second derivatives of the orientation. On the other hand, it also leads to sweeping graceless motion because little force can be applied to the object, since force is proportional to acceleration, which was minimized.

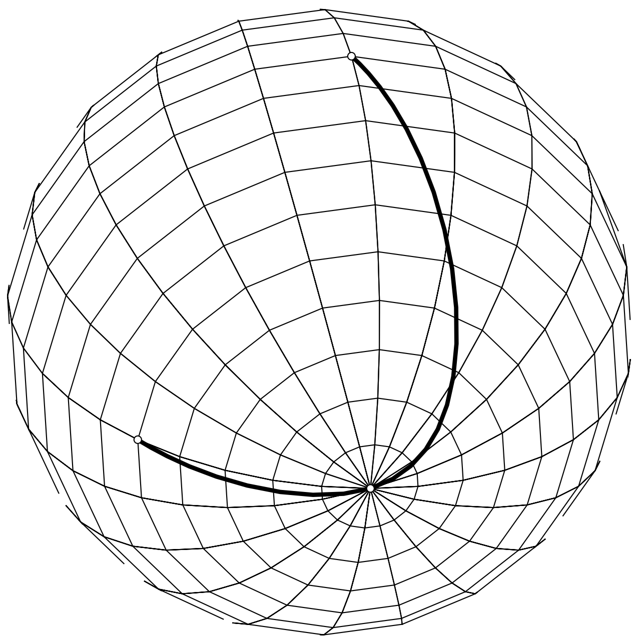

Figure 2.3: Motion path created by minimizing rotational acceleration; the white dots are key frames. □

Tangential Acceleration

In the previous experiment (minimization of rotational acceleration) we did not constrain the angular velocity or orientation of the objects at all beyond forcing them to meet the conditions of the key frames at the appropriate times. Those were treated as hard constraints, and were required to be met at all times. To enforce this condition, the optimizer could not vary these quantities.

We allowed, however, the optimizer to generate non-unit quaternions and just normalized them before evaluating the objective function. This implies a soft constraint that the quaternions be unit length, that is, that

$$q_i(t)q_i(t) = 1. \tag{2.4}$$

Optimizers perform better with constraint function that are bounded below, so should we choose to force our optimizer to maintain this constraint, we will implement it as

$$(q_i(t)q_i(t) - 1)^2 = 0. \tag{2.5}$$

Since the initial conditions given to the optimizer contain only unit quaternions and the optimizer typically does not change the values of the quaternions by large amounts, we do not encounter the meaningless quaternion $(0, 0, 0, 0)$. As a consequence, the projection of an optimizer solution onto the constraint manifold is straightforward.

The normal to this constraint manifold $g(q)$ is also straightforward:

$$g(q): \; (q_i(t)q_i(t) - 1)^2 = 0 \tag{2.6}$$

and

$$N_i(g) = \nabla_i g = 4q_i(t)(q_j(t)q_j(t) - 1). \tag{2.7}$$

This is just parallel to $q(t)$, so the gradient of the constraint manifold is in the direction of the quaternion representing the orientation of the body.

To project the angular acceleration of the body into the tangent plane of the constraint manifold, we remove the portion of the acceleration that is in the direction of the normal to it. That projection is

$$\frac{D^2}{dt^2}q(t) = \ddot{q}(t) \setminus \nabla g. \tag{2.8}$$

The "without" operator, "\" expands to

$$(a \setminus b)_i = a_i - b_i \frac{a_j b_j}{b_l b_l}. \tag{2.9}$$

For the case of the covariant angular acceleration, we get

$$\frac{D^2}{dt^2}q(t) = \ddot{q}_i - q_i \frac{\ddot{q}_j q_j}{q_l q_l}. \tag{2.10}$$

Replacing $\ddot{q}$ with $\frac{D^2 q(t)}{dt^2}$ in Equation 2.3 and minimizing the corresponding objective function, we found that minimizing the tangential acceleration resulted in motion paths very similar to those we found when minimizing the ordinary acceleration.

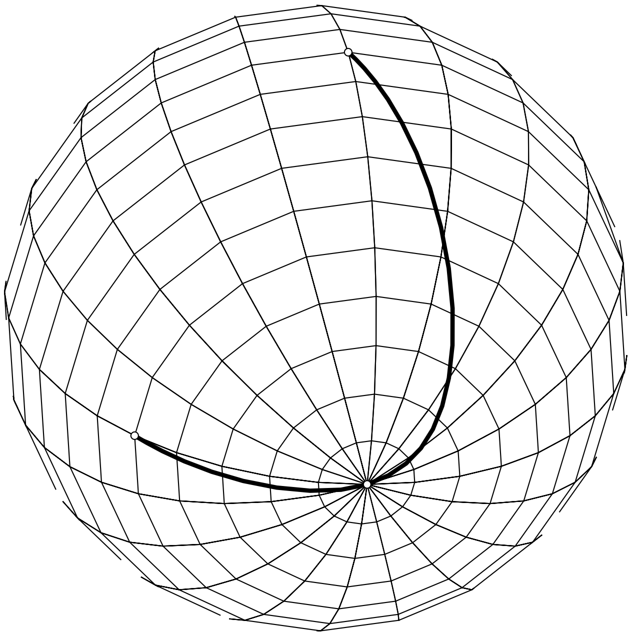

Figure 2.4: Motion path created by minimizing tangential rotational acceleration; the white dots are key frames. □

Comparing Figures 2.3 and 2.4, we can see that, at least in this example, the paths generated by minimizing tangential acceleration and regular acceleration are very similar. Contrast Figure 2.2, the path generated by minimization of kinetic energy. The animation generated by that path is quite different, far less smooth, and quite a bit less attractive.

2.2.2 Multiple-Body Objective Functions

The objective functions we considered for multiple-body cases are fairly similar to the ones for single-body examples, with the exception of the last four (acceleration and tangential acceleration of specific points). The formulae differ, though, as do the effects.

For all the multiple-body objective functions, we shall minimize a quantity of the form

$$E = \sum_{b=1}^{nbodies} \int_{t_{min}}^{t_{max}} f_b(t)dt. \tag{2.11}$$

$f_b(t)$ is some function describing some information about body b. For each different objective function, f will be different, but the sum and integral will remain.

Kinetic Energy

The derivation for the Kinetic Energy of children of a root body can be found in Appendix B1. While much more complicated than that of a single body, it is still basically a velocity squared measure, so the result of minimization of it yielded a very stiff-looking motion with sudden accelerations right around the key frames in order to get to the right "average" (constant) angular velocity.

Acceleration and Tangential Acceleration

Minimizing angular acceleration, with or without the normal component, produced similar effects as for the one-body case: broad, sweeping motions that were extremely smooth, but exaggerated and not very attractive.

Acceleration of Specific Points

The rotational acceleration of a body does not depend on the geometry of the body or of the attachment scheme. A good objective function to minimize, however, ought to include some information about the geometry. After all, a spinning dumbbell shape ought to behave differently if one of the ends were cut off. The kinetic energy scheme was intended to use this information, but it had other flaws that are difficult to overcome.

A function that has some of these desirable characteristics is the ordinary spatial acceleration of the center of mass of the object. For a single body rotating around its center of mass, the acceleration is zero, and thus not very interesting, but a center of mass of a body rotating around some other point, as of the secondary bodies of an articulated figure, might produce interesting motions.

Other points than the center of mass can be chosen. We implemented the acceleration of the center of mass; the equations corresponding to choices of other points can easily be seen from the equations for acceleration of the center of mass.

For an articulated figure as shown in Figure 2.2, the position of the center of mass of the secondary body is

$$x^1_{cm}(t) = x^0_{cm}(t) + q^0(t)\ a\ (q^0)^{-1}(t) + q^0(t)q^1(t)\ b\ (q^1)^{-1}(t)(q^0)^{-1}(t) \tag{2.12}$$

where x^0_{cm} is the center of mass of the root body, x^1_{cm} is the center of mass of the secondary body, q^0 is the quaternion representing the orientation of the root body, q^1 is the quaternion representing the orientation of the secondary body, a is the vector from the center of mass of the root body to the attachment point of the secondary body, and b is the vector from the attachment point of the secondary body to the center of mass of the secondary body. If a point other than the center of mass of the secondary body were to be chosen, then b would simply be the vector to that point from the attachment point. The multiplication in this (and the next few) equation(s) is quaternion multiplication. (See Appendix C.) All 3-vectors are replaced by $(0, v_0, v_1, v_2)$ for this purpose.

Taking the first two time derivatives of Equation 2.11, we get

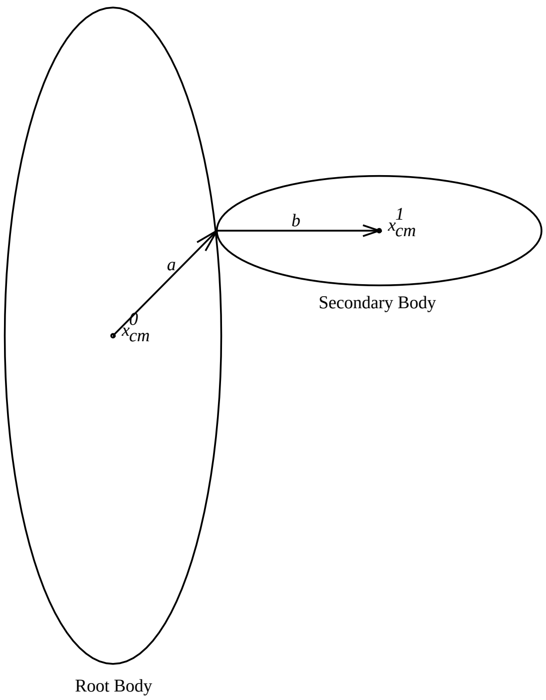

Figure 2.5: An articulated figure showing the vectors used in Equations 2.12 through 2.15. □

$$\begin{aligned}
\frac{d}{dt}(x^1_{cm}) &= \dot{x}^0_{cm} + \dot{q}^0\ a\ (q^0)^{-1} + q^0\ a\ (\dot{q}^0)^{-1} + \\
&\quad \dot{q}^0 q^1\ b\ (q^1)^{-1}(q^0)^{-1} + q^0 \dot{q}^1\ b\ (q^1)^{-1}(q^0)^{-1} + \\
&\quad q^0 q^1\ b\ (\dot{q}^1)^{-1}(q^0)^{-1} + q^0 q^1\ b\ (q^1)^{-1}(\dot{q}^0)^{-1} \qquad (2.13)\\
\frac{d^2}{dt^2}(x^1_{cm}) &= \ddot{x}^0_{cm} + \ddot{q}^0\ a\ (q^0)^{-1} + \dot{q}^0\ a\ (\dot{q}^0)^{-1} + q^0\ a\ (\ddot{q}^0)^{-1} + \\
&\quad \ddot{q}^0 q^1\ b\ (q^1)^{-1}(q^0)^{-1} + q^0 \ddot{q}^1\ b\ (q^1)^{-1}(q^0)^{-1} + \\
&\quad q^0 q^1\ b\ (\ddot{q}^1)^{-1}(q^0)^{-1} + q^0 q^1\ b\ (q^1)^{-1}(\ddot{q}^0)^{-1} + \\
&\quad 2(\dot{q}^0 \dot{q}^1\ b\ (q^1)^{-1}(q^0)^{-1} + \dot{q}^0 q^1\ b\ (\dot{q}^1)^{-1}(q^0)^{-1} + \dot{q}^0 q^1\ b\ (q^1)^{-1}(\dot{q}^0)^{-1} + \\
&\quad q^0 \dot{q}^1\ b\ (\dot{q}^1)^{-1}(q^0)^{-1} + q^0 \dot{q}^1\ b\ (q^1)^{-1}(\dot{q}^0)^{-1} + q^0 q^1\ b\ (\dot{q}^1)^{-1}(\dot{q}^0)^{-1}). \qquad (2.14)
\end{aligned}$$

We minimized

$$E = \sum_{b=1}^{nbodies} \int_{t_{min}}^{t_{max}} \frac{d}{dt}(x^b_{cm}) \cdot \frac{d}{dt}(x^b_{cm}) dt. \qquad (2.15)$$

This worked surprisingly well, though with a few drawbacks. The global minimum for this problem was such that the centers of mass of secondary bodies were moved to the center of rotation of the entire body. To prevent this, we added a constraint that secondary bodies could not rotate more than a given angle from their initial positions.

With the constraint, the resulting motion was very interesting. The secondary bodies moved ahead of the root body as to appear to "anticipate" the rotation of the root body. They also swung around in nice smooth paths.

Another drawback of this scheme was that the secondary bodies were oriented randomly with respect to the twist around the axis between the center of mass and the attachment point. That makes sense; only the position of the center of mass is used in calculating the objective that was minimized.

Not surprisingly, this same approach can be performed using covariant acceleration rather than normal acceleration. We found that the motion paths looked a little better when using covariant acceleration. The difference was not large.

Acceleration of All Points

In order to improve the previous approach, we considered computing the acceleration of another point on the body and adding that in to the objective function. This would prevent the random orientation problem, assuming that this third point was not colinear with the attachment point and the center of mass.

If one point is good and two points are better, why stop at two? Why not add up the acceleration of all the points in the body? This can be done; it is just the integral over the volume of the body of the acceleration of each point.

Computing a volume integral in the inner loop of an optimization problem would lead to excessive computation times. Fortunately, a large part of the integral can be computed analytically, at least for simple shapes.

For bodies attached to the root body, this objective function can be represented as

$$E = \sum_{b=1}^{nbodies} \int_{t_{min}}^{t_{max}} \int_{V_b} \ddot{x}^b(t,u) \cdot \ddot{x}^b(t,u) dV dt \tag{2.16}$$

where $x^b(t,u)$ is the position of a point within body b. If $q^0(t)$ is the orientation (represented as a quaternion) of the root body, $q^1(t)$ is the orientation of body 1 relative to body 0, a is the vector between the geometric center of body 0 and the attachment point between the two bodies (represented in body 0's coordinate frame), and u is the vector from the attachment point to x^1 (represented in body 1's coordinate frame) (see Figure 2.6) we get

$$x^1(t,u) = x^0_{gc}(t) + q^0(t)\ a\ (q^0)^{-1}(t) + q^0(t)q^1(t)\ u\ (q^1)^{-1}(t)(q^0)^{-1}(t). \tag{2.17}$$

We can factor this into the form

$$x^1_i(t,u) = A_i(t) + B_{ij}(t)u_j. \tag{2.18}$$

Taking the first and second time derivatives of $x^1(t,u)$, we get

$$\dot{x}^1(t,u) = \dot{A}_i(t) + \dot{B}_{ij}(t)u_j \tag{2.19}$$

$$\ddot{x}^1(t,u) = \ddot{A}_i(t) + \ddot{B}_{ij}(t)u_j \tag{2.20}$$

because only A_i and B_{ij} contain any time-varying terms.

Substituting this in the innermost integral in equation 2.14,

$$\int_{V_b} (\ddot{x}^b(t,u))^2 dV = \int_{V_b} (\ddot{A}_i(t) + \ddot{B}_{ij}(t)u_j)^2 dV \tag{2.21}$$

$$= \int_{V_b} (\ddot{A}_i(t)\ddot{A}_i(t) + 2\ddot{A}_i(t)\ddot{B}_{ij}(t)u_j + \ddot{B}_{ij}(t)u_j\ddot{B}_{ik}(t)u_k) dV \tag{2.22}$$

Since

$$\int_{V_b} dV = V_b, \tag{2.23}$$

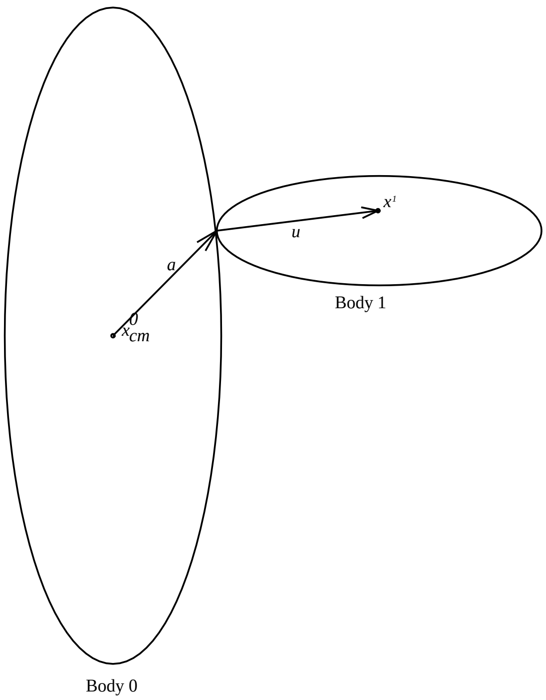

Figure 2.6: An articulated figure showing the vectors used in Equations 2.17 through 2.42. □

and

$$\int_{V_b} u_i dV = V_b (x^b_{gc})_i, \tag{2.24}$$

where V_b is the volume of body b and x^b_{gc} is the geometric center of body b, and

$$\int_{V_b} u_i u_j dV = I_{ij}/m, \tag{2.25}$$

where I_{ij} is the second moment of mass distribution of the object, and m is the mass of the object, this simplifies to

$$\int_{V_b} (\ddot{x}^b(t,u))^2 dV = V_b \ddot{A}_i(t)\ddot{A}_i(t) + 2V_b \ddot{A}_i(t)\ddot{B}_{ij}(t)(x^b_{gc})_j + \frac{1}{m}\ddot{B}_{ij}(t) I_{jk} \ddot{B}^T_{ki}(t). \tag{2.26}$$

$\ddot{A}_i$ and $\ddot{B}_{ij}$ are just functions of quantities already known and can be shown to be

$$\begin{aligned} \ddot{A}_i \quad = \quad & (\ddot{x}^0_{gc})_i + \kappa_{ijk}\vec{\kappa}_{jlm}\ddot{q}^0_l a_m (q^0_k)^{-1} + \\ & 2\kappa_{ijk}\vec{\kappa}_{jlm}\dot{q}^0_l a_m (\dot{q}^0_k)^{-1} + \kappa_{ijk}\vec{\kappa}_{jlm} q^0_l a_m (\ddot{q}^0_k)^{-1} \end{aligned} \tag{2.27}$$

and

$$\ddot{B}_{ij} = D^{02}_{ij} + 2D^{11}_{ij} + D^{20}_{ij} \tag{2.28}$$

such that

$$
\begin{aligned}
D^{02}_{ij} &= \kappa_{ijk}\vec{\kappa}_{jlm} C_l \ddot{C}^{-1}_k && (2.29)\\
D^{11}_{ij} &= \kappa_{ijk}\vec{\kappa}_{jlm} \dot{C}_l \dot{C}^{-1}_k && (2.30)\\
D^{20}_{ij} &= \kappa_{ijk}\vec{\kappa}_{jlm} \ddot{C}_l C^{-1}_k && (2.31)\\
C_i &= \kappa_{ijk} q^0_j q^1_k && (2.32)\\
\dot{C}_i &= \kappa_{ijk} \dot{q}^0_j q^1_k + \kappa_{ijk} q^0_j \dot{q}^1_k && (2.33)\\
\ddot{C}_i &= \kappa_{ijk} \ddot{q}^0_j q^1_k + 2\kappa_{ijk} \dot{q}^0_j \dot{q}^1_k + \kappa_{ijk} q^0_j \ddot{q}^1_k && (2.34)\\
C^{-1}_i &= \kappa_{ijk} (q^1_j)^{-1} (q^0_k)^{-1} && (2.35)\\
\dot{C}^{-1}_i &= \kappa_{ijk} (\dot{q}^1_j)^{-1} (q^0_k)^{-1} + \kappa_{ijk} (q^1_j)^{-1} (\dot{q}^0_k)^{-1} && (2.36)\\
\ddot{C}^{-1}_i &= \kappa_{ijk} (\ddot{q}^1_j)^{-1} (q^0_k)^{-1} + 2\kappa_{ijk} (\dot{q}^1_j)^{-1} (\dot{q}^0_k)^{-1} + \kappa_{ijk} (q^1_j)^{-1} (\ddot{q}^0_k)^{-1}. && (2.37)
\end{aligned}
$$

Minimizing E in equation 2.14 produced motion pretty much as was to be expected. The orientation of the body along the axis from the connection point to the center of mass was no longer random, but instead was with the smallest (and furthest extended) part of the body. This leading was synchronized automatically with the motion of the body through its entire path. See Chapter 3 for an example of this technique in action.

Tangential Acceleration of All Points

Instead of integrating the square of the acceleration, we can do the same with the tangential acceleration. Our innermost integral becomes

$$\int_{V_b} (\ddot{x}^b(t,u) \setminus \nabla g)(\ddot{x}^b(t,u) \setminus \nabla g) dV. \tag{2.38}$$

Since

$$\ddot{x}_i = \ddot{A}_i + \ddot{B}_{ij} u_j, \tag{2.39}$$

and because the $\setminus$ operator distributes,

$$(\ddot{x} \setminus \nabla g)_i = (\ddot{A} \setminus \nabla g)_i + \ddot{B}_{ij} u_j \setminus \nabla_i g. \tag{2.40}$$

Letting N_i be $\nabla_i g$, we can define

$$\hat{A}_i = (\ddot{A} \setminus N)_i = \ddot{A}_i - \frac{\ddot{A}_j N_j}{N_k N_K} N_i \tag{2.41}$$

and

$$\hat{B}_{ij} = \ddot{B}_{ij} \setminus N_i = B_{ij} - \frac{B_{lj} N_l}{N_k N_k} N_i \tag{2.42}$$

and then to replace the standard acceleration with the covariant acceleration in the innermost integral, we just substitute $\hat{A}$ for $\ddot{A}$ and $\hat{B}$ for $\ddot{B}$.

For multiple bodies, the constraint function $g_n(q^0, q^1, ..., q^n)$ is the requirement that each quaternion representing rotation be unit length. The gradient of that function is parallel to each of the individual quaternions. If other constraints are included (for example, angle limits) g may be modified to account for them.

Minimization of covariant acceleration of all points differs from regular acceleration only slightly, but it seems to converge faster and produces slightly smoother motion.

2.3 Converting to an Optimization Problem

We have considered many quantities to minimize in order to generate motion paths for objects in computer animation. A few details need be considered before performing this minimization. Those details are variables and constraints, which are primarily details of setting up optimization problems. We shall also briefly consider solving the optimization problem in the next section.

2.3.1 Hard and Soft Constraints

We consider two types of constraints. The first type are those we call "hard constraints," constraints that must never be violated. The second type, "soft constraints," are constraints that must be met (or possibly almost met) by the final solution of the optimization problem, but may be violated during the search for the solution, including, possibly, by the initial conditions.

We treat as hard constraints that information which is fixed at the key frames. We require that initial conditions for our implementation meet these constraints. This makes sense because in order to specify key frame information, the user must already know it.

If, however, a user were to want to specify a key frame orientation or angular velocity to be "approximately" as given, we would need to implement that as a soft constraint, since any value within a range is acceptable. We have not yet implemented keyframe constraints of this form.

Soft constraints include any sort of conditions whose violation would not make computation of the energy of such a state impossible. The major soft constraint we use is that all quaternions must be of unit length. We have implemented this constraint by settin up our objective functions always to evaluate a state correctly even if quaternions are non-unitary. We do not just normalize them, because computation of gradients must be done at the actual point in the space we are searching, not at an equivalent point. This scheme works fine as long as the solver stays away from the quaternion $(0, 0, 0, 0)$, and we have found that typical step sizes in an optimizer are small enough that it has never become close to that point.

Note that the unitary quaternion constraint is a non-linear constraint. Since most solvers will require the ability to evaluate the objective function off this constraint manifold, objective functions must be able to be evaluated with non-unit quaternions. For this purpose, normalization is adequate, but we find that, in general, it is not necessary. The reason for this is that we are using quaternions to represent rotations only. The rotation of a vector a by a quaternion q is

$$\hat{a} = q\ a\ q^{-1}. \tag{2.43}$$

If q is of non-unit length, we can replace it with mQ where m is the magnitude of the quaternion and Q is a unit quaternion in the same direction as q. The inverse of non-unit quaternions is the normal inverse (negation of the last three components) divided by the square of the length of the quaternion, so

$$\hat{a} = (mQ)\ a\ (mQ^{-1})/(q \cdot q) \tag{2.44}$$

Since

$$\begin{aligned} q \cdot q &= m^2, && (2.45)\\ \hat{a} &= (mQ)\ a\ (mQ^{-1})/m^2 && (2.46)\\ \hat{a} &= Q\ a\ Q^{-1}. && (2.47) \end{aligned}$$

That means that we can use non-unit quaternions as long as we stay away from $m = 0$.

Other soft constraints that might be valuable include limits on rotations at connections and limits on angular velocity. We have not implemented these constraints, because solving the so-produced problems usually requires a much more complicated solver than we currently have in place.

2.3.2 Subintervals of Keyframe Intervals

An optimization problem consists of three components: an objective function to be minimized, a set of constraints (perhaps the null set), and a space in which to search for the minimum to the objective function.

The objective functions described earlier were all actually functionals producing a scalar value as a function of a set of time-varying functions. Those time-varying functions are the quaternions that represent the orientation of the objects, generally called $q^0(t), q^1(t), ..., qn(t)$ in the text above. Thus, the space in which we need to minimize our objective functions is a space of functions. Numerical optimizers (solvers), however, require that the space in which they must search be a space of numbers. In order to use a numerical solver, we need to convert the space

$$S = (q^0(t) \times q^1(t) \times ... \times q^n(t)) \tag{2.48}$$

to a space of numbers.

We did this by approximating each $q(t)$ as four piecewise cubic polynomials, one for each component of q. As the number of pieces in this representation becomes very large, our approximation becomes able to represent any continuous function $q(t)$ to within smaller and smaller errors.

A byproduct of this approximation is that we create a new set of important times which are the endpoints of the cubics. Currently, we create these times *a priori* under user control. Optimally, the system would create them for us, but we have not implemented such a scheme yet. We call the times bounded by the endpoints of the cubics *subintervals*.

We chose cubics because they seemed flexible enough to approximate the functions we expected would be solutions to the optimization problem and because their derivatives can be computed analytically. If we had used simpler functions than cubics, we would have needed a larger number of subintervals to produce adequate solutions.

The cubic polynomials still need to be parameterized in order to reduce our space of functions into a space of numbers. The obvious parameterization of a cubic polynomial is $f(t) = a\,t^3 + b\,t^2 + c\,t + d$ where a, b, c, and d are real numbers. At the key frames, however, we know the value of the function and the value of its first derivative because angular velocity can be computed from a quaternion representation of orientations thus:

$$\begin{pmatrix} 0 \\ \omega \end{pmatrix} = 2\dot{q}(t)q^{-1}(t) \tag{2.49}$$

The parameterization of cubics suggested by the knowledge of the value of the function and its first derivative at the endpoints of the interval it covers is the Hermite polynomial. The derivation of the Hermite polynomial is in Appendix B2. The Hermite polynomial is a matrix spline, which means that it can be represented as a constant matrix premultiplied and postmultiplied by vectors thus:

$$q(t) = (t^3\ t^2\ t\ 1) \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} q^0 \\ q^1 \\ \frac{1}{2}\omega^0 q^0 \\ \frac{1}{2}\omega^1 q^1 \end{pmatrix}. \tag{2.50}$$

ω^0 is the angular velocity at the beginning of the interval, that is, at $t = 0$, and ω^1 is similar for $t = 1$. $q^0 = q(0)$ and $q^1 = q(1)$. The whole equation is normally written

$$q(t) = T\ H\ K \tag{2.51}$$

with the three factors corresponding to the three factors in the previous equation. H is usually called the *magic matrix* for the Hermite polynomial and K is called the *knot vector*. The first time derivative is simple to compute and is

$$\dot{q}(t) = T\ \dot{H}\ K \tag{2.52}$$

with

$$\dot{H} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 6 & -6 & 3 & 3 \\ -6 & 6 & -4 & -2 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.53}$$

Note that $\dot{H}$ is not strictly the time derivative of H, but is the version of the magic matrix that is used to compute the time derivative of the Hermite polynomial.

The second time derivative is also useful for some of the objective functions we investigated, and is

$$\ddot{q}(t) = T\ \ddot{H}\ K \tag{2.54}$$

such that

$$\ddot{H} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 12 & -12 & 6 & 6 \\ -6 & 6 & -4 & -2 \end{bmatrix} \tag{2.55}$$

This parameterization leads to a nice feature of our optimization scheme. The space of functions of quaternions is parameterized by the components of the knot vector, which is simply a combination of the values of the quaternion itself and of the angular velocity of the bodies at a set of times. That is, the variables in our optimization problem are just angular velocities and orientations of the bodies at a set of times.

2.4 Solving the Optimization Problem

In order to solve optimization problems consisting of one of the objective functions considered above, a set of constraints, and the space described in the previous section, we must choose the number of subintervals between each key frame. We also must choose whether or not to specify the angular velocity of some bodies at the key frames. The latter choice will depend upon the problem we want to solve. The former choice, however, is an interesting tradeoff. More subintervals will allow for a perhaps better solution, though more than one subinterval between frames of an animation seems excessive, since information between subinterval endpoints can be computed by evaluating the appropriate interpolating cubic polynomial. More subintervals increase the dimension of the optimization problem, slowing down the solution. The number of dimensions of the problem is

$$n = 7n_{bodies}\ n_{subintervals} + 3n_{\omega} \tag{2.56}$$

where n_{bodies} is the number of bodies in the scene, $n_{subintervals}$ is the number of subinterval endpoints used (excluding key frames,) and n_{ω} is the number of key frames at which the angular velocity is unconstrained. n_{ω} is counted separately for each body. The reason for the factor of seven in the first term is that at each subinterval endpoint that is not a key frame, the four components of the quaternion representing orientation and the three components of angular velocity vary.

Once all these decisions are made, a numerical optimizer can be used to find the minimum of the energy function. This will produce a specification of the orientation and angular velocity of each body at the endpoints of each subinterval and at the key frames. The Hermite polynomial described above is used to interpolate these values within the subinterval, allowing for straightforward evaluation of $q(t)$ and $\omega(t)$ for each body.

2.4.1 Numerical Constrained Optimization

Many numerical constrained optimization algorithms exist, with differing speeds and other benefits. The two simplest solvers are gradient descent and conjugate gradient descent. More complicated methods include projected Lagrangian methods and sequential quadratic programming (SQP) methods.

All methods require evaluation of the gradient of the objective function; some require the computation of the Hessian matrix as well. If the gradient cannot be computed symbolically, which is often the case, it must be computed numerically. The same is true for higher derivatives.

In their notes about conjugate gradient methods, Gill et al. [Gill] claim that a reasonable number of iterations for that method is $5n$ where n is the number of dimensions in the problem. The bulk of the computation at each step is the computation of the gradient. If this needs be approximated numerically, it is an $\mathcal{O}(n)$ operation. That suggests that the average case performance of Conjugate Gradient is $\mathcal{O}(n^2)$. If the objective function to be minimized is complicated enough that the numerical gradient takes a fair bit of time to compute, as do many of the ones we consider here, a large number of dimensions in which to optimize will create prohibitively large computation costs for these problems.

Better solvers have somewhat better asymptotic time complexity, but they are not a panacea. Complicated methods are hard to implement; the subtleties of programming an SQP solver are notorious. Several such solvers are commercially available; see [Minos] and [NAG] for details of two of them.

The faster algorithms have a hidden drawback. The implementation of some of our objective functions is not difficult, but their evaluation is often difficult to do by hand, so unit-testing them is tricky. If an error in the implementation of the objective function occurs, an optimizer might get very strange results. This is exacerbated by the fact that optimizers sometimes need to evaluate the objective function at places in the search space that are not on the constraint manifold and might be far enough away that such input conditions were never considered. A hand-written simple solver can be programmed to display much useful data about its progress and, therefore, can strongly reduce implementation time.

The previous effect is not just limited to implementation errors in the objective function. It is quite easy not to notice that a correctly-implemented function might produce silly answers for inputs that are unexpected. An optimizer will often find those inputs. When this happens, the optimizer will fail in strange and mysterious ways. Typical error messages such as "superbasic limit exceeded" will not generally be helpful to any but the most expert user. A major caveat for one designing objective functions is to ensure that the objective function yields reasonable answers anywhere near the constraint manifold. This might require causing it to produce answers that have very little to do with the original problem, and it might well be tricky to pick answers that make sense and do not confuse the optimizer. For example, just setting the objective function to a large number off the constraint manifold will almost certainly cause the optimizer to produce incorrect results, or no results at all.

For the objectives described above, only the neighborhood of zero for a quaternion is likely to cause this sort of error, but if further constraints are required, care must be taken near the boundaries of the feasible space created by the additional constraints. In particular, if a constraint-caused search space boundary passes through the zero quaternion, an optimizer is likely to encounter problems.

One of two general approaches for enforcing constraints is usually used. For soft constraints, the "penalty method" increases the value of the objective function due to violation of constraints. This penalty function must be smooth near the constraint boundary, though what degree of smoothness is needed depends on the individual solver and problem. This method has some drawbacks. One is that the constraint will not be exactly met. The optimal point of the constrained problem will be on the boundary where the penalty function evaluates to zero. As a consequence, the solution found by penalty-method optimizers will be slightly beyond the edge of the feasible search space. One cannot ensure that the solution point is a tiny bit beyond the edge by increasing the size of the penalty; that does not work. (See [Gill] for details.) In general, penalty methods also require the adjustment of penalty weights, which can be a very time-consuming and unrewarding process.

The other general approach, for enforcing hard constraints, is the method of Lagrange multipliers. The objective function is augmented by adding a term that is the dot product of the constraint function (which produces the size of the violation of the constraint) and a set of free parameters, which are to be maximized. See [Gill] for much more detail on Lagrange multipliers. A major drawback of this approach is that if the initial estimates of the multipliers are not accurate, most solvers will be unable to find a minimum. In many cases, these estimates do not have to be very inaccurate for this to happen. Selection of the initial estimates for Lagrange multipliers can sometimes be done automatically, but an automatic approach will sometimes fail; if an inexperienced user uses such an automatic scheme, the failure may be very mysterious and appear similar to an implementation or user error.

2.4.2 Some Efficiency Considerations

Because the computation time taken by an optimizer is strongly dependent on the number of dimensions of the search space, some tricks are useful for speeding up the optimization. In some geometries, we can find useful symmetries. If the answer to a problem is known to be symmetrical, then only one half of the problem need be solved, or perhaps the symmetry can be coded into the objective function, reducing the dimension of the search space.

Most of the objective functions described here produce contributions from each body in the scene. Some bodies' contributions will be independent from others'; in that case, a separate invocation of the optimizer can be used to determine the optimal motion for each of the independent bodies. Other times, these contributions will be almost independent from each other. This will most typically happen if one body has a large-valued objective function and the other's is very small. Again, these objects' motions are likely to be able to be optimized independently, saving large amounts of computation time.

If a symmetry or independent subproblem is almost present, but not quite, solving the reduced problem can often be used to find initial conditions for a run of the full problem. Optimizers converge much more quickly if their initial conditions are close to a local minimum.

One other characteristic of problems with symmetries that might cause somewhat surprising results follows from numerical optimization's only finding local minima, not global minima. Frequently, several local minima exist near the initial conditions. An optimizer will converge to one of them, but not necessarily to the same one given small perturbations of the initial conditions. Symmetries most often seem to be the cause of effects of this sort; an obvious initial path might well be a saddle point in the search space, causing slight tinkers to the initial conditions to lead to large changes in the results.

In general, optimization is not a fast approach. Using a single HP 700 workstation and a simple conjugate gradient optimizer, one optimization run for two bodies and ten non-key frames took between 26 minutes and nine hours, depending on the objective function chosen. Timings for one example run are as follows:

Acceleration of one point	64 minutes
Covariant Acceleration of one point	26 minutes
Acceleration of all points	524 minutes
Covariant Acceleration of all points	438 minutes

Chapter 3

Results and Conclusions

3.1 Surfboard Man

The example model used figure 2.1 is composed of three ellipsoids. The root body is the big ellipsoid in the middle, and the secondary bodies are the two smaller ellipsoids on the sides. We call this object "surfboard man" for obvious reasons.

In the next few figures, surfboard man is rotating counterclockwise as viewed from the top. The figures show twelve small versions of him in different orientations. The frames are ordered as one might expect, left to right, then top to bottom. The first image and the seventh image in each figure are key frames. A third key frame and thirteenth image is not drawn, but is identical to the first one. In each example, surfboard man rotates one full revolution between each pair of key frames. The secondary bodies (arms) are constrained only at the key frames. Their angular velocities are constrained to be zero at the first and thirteenth frame, but allowed to vary in the seventh frame.

Figures 3.1 and 3.2 are a top view and side view of surfboard man with the acceleration of all the points in all three bodies minimized. Note how the arms first lag behind the rotation, then anticipate it.

Figures 3.3 and 3.4 are a top view and side view of surfboard man with the tangential acceleration of all the points in all three bodies minimized.

3.2 Future Work

3.2.1 Flux and Air Resistance

Especially in cases in which bodies are both translating and rotating, we might want objects to align themselves so that their longest axis is parallel to their direction of travel, or their shortest ones are perpendicular to it. One way to achieve this goal is to add air resistance to the objective function. That might not, however, be the reason that we want this alignment. We might just like the lines created by this orientation; the air resistance term might not be significant, either. In that case, we might choose to minimize the air "flux" of the body, that is, the cross-sectional area of the body in the direction that it is translating, integrated, as usual, over time. Most likely, we would just want flux to be part of the objective function, with smoothness criteria as part of the objective, also.

We have not pursued this direction, but it might produce interesting and attractive motions. Ballerinas, for example, often attempt to position their limbs to create the appearance of straight lines and to be parallel to their direction of travel.

3.2.2 Gravity and Obstacles

We did not pursue much interaction between moving bodies and their environment. [Witkin] found that this was a very rewarding direction to pursue; a complete system ought to be able to include these effects.

3.2.3 Something done with Kinetic Energy

We discovered that minimization of kinetic energy by itself produced uninteresting motion. We still feel, however, that some physical quantities ought be either part of an objective function, or can be useful for some applications. We have not discovered how to do this, but have not entirely ruled out the possibility that kinetic energy can be effectively used somehow.

3.2.4 Subdivision Criteria

As we constructed our system, subintervals were created *a priori.* Animators, in general, will not know how to do this and ought not need to know. The system ought to choose these for them automatically. One approach that we have considered is progressive refinement. The system will automatically choose some number of intervals via a heuristic, and then solve the optimization problem. Then, for each interval, a second run can be done, having split the interval into two or more pieces. If the difference between the two paths so generated is sufficiently small, the algorithm will terminate, but if not, it will continue to subdivide intervals in which significant change occurs. An alternative approach is to try to predict which intervals are likely to need subdivision without rerunning the optimization. One measure might be the curvature of the motion path in the interval. A very curved path might be best replaced with two cubic pieces. Another measure might be found by looking at the coefficients of the cubics. Wiggles in the computed motion path might suggest that more resolution is necessary in the area around the wiggles.

3.3 Conclusions

[Witkin] considered using optimization to plan motion paths for complex objects within environments. They found that their technique produced very interesting animation, particularly if constraints were chosen carefully.

We have found that the choice of objective function within this scheme is important as well. Minimizing the acceleration of all points in a body produces motion that makes the bodies appear to anticipate their next motions as well as motion that is very smooth.

Tangential acceleration appears to produce optimization problems that converge more quickly and usually gives slightly better results. As the constraint manifold becomes more complicated than the ones we used, we expect the differences to become larger.

Whether or not this sort of scheme will move into everyday use depends mostly on whether or not constrained optimization can be made to work automatically with no intervention or tinkering by a user. Currently, we feel that optimizers do not meet this criterion. Numerical optimization is also very slow. We expect that it is faster to produce animation by optimization than it is by hand, but not enough so that the approach is interactive yet. As computers get faster and optimizers get more user-friendly, especially very good solvers, optimized computer-generated motions may be more frequently used to design animation.

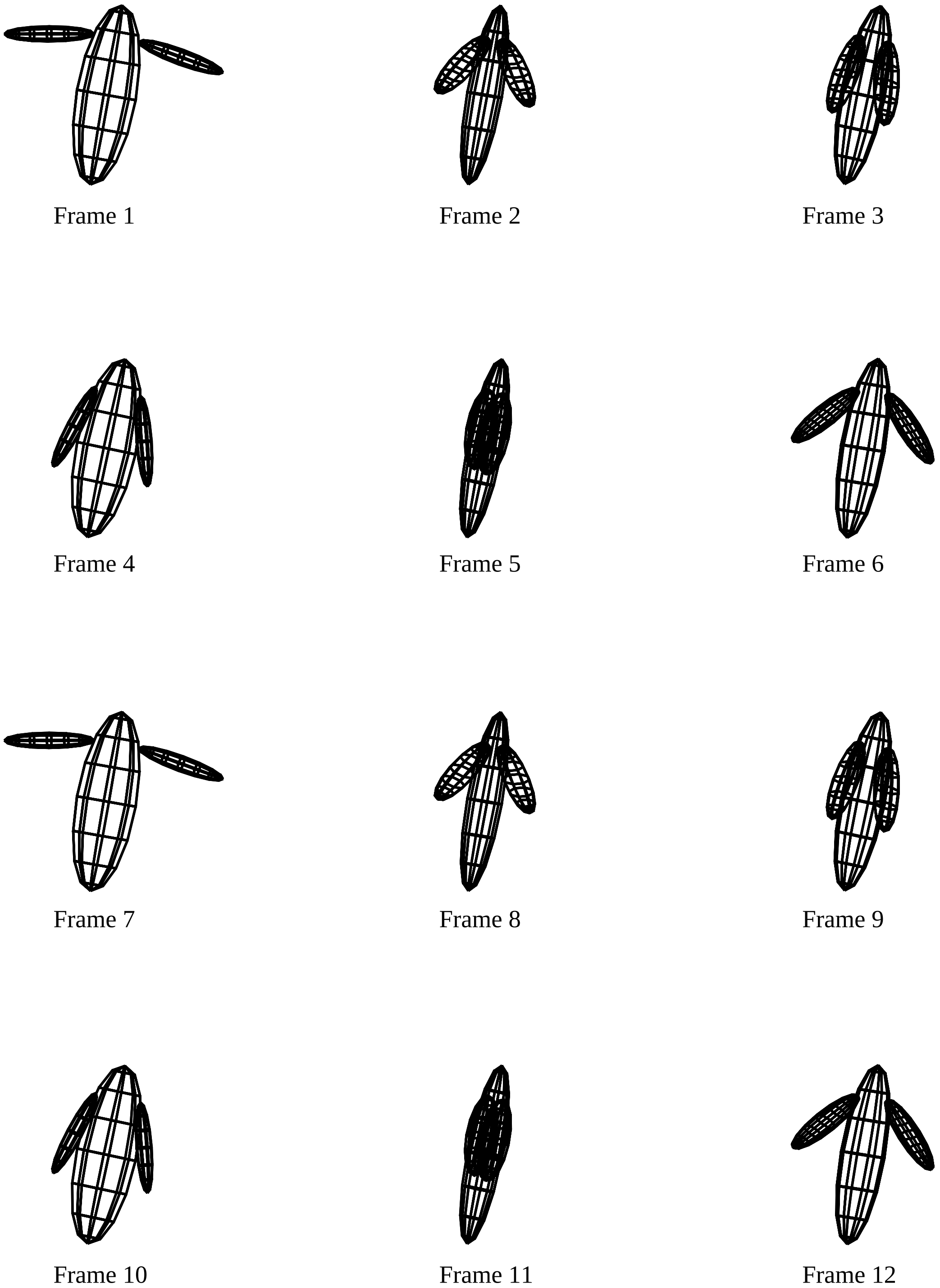

Figure 3.1: Motion created by minimization of acceleration of all points on all bodies. Images 1, 7, and 13 (omitted, but identical to 1) are key frames. □

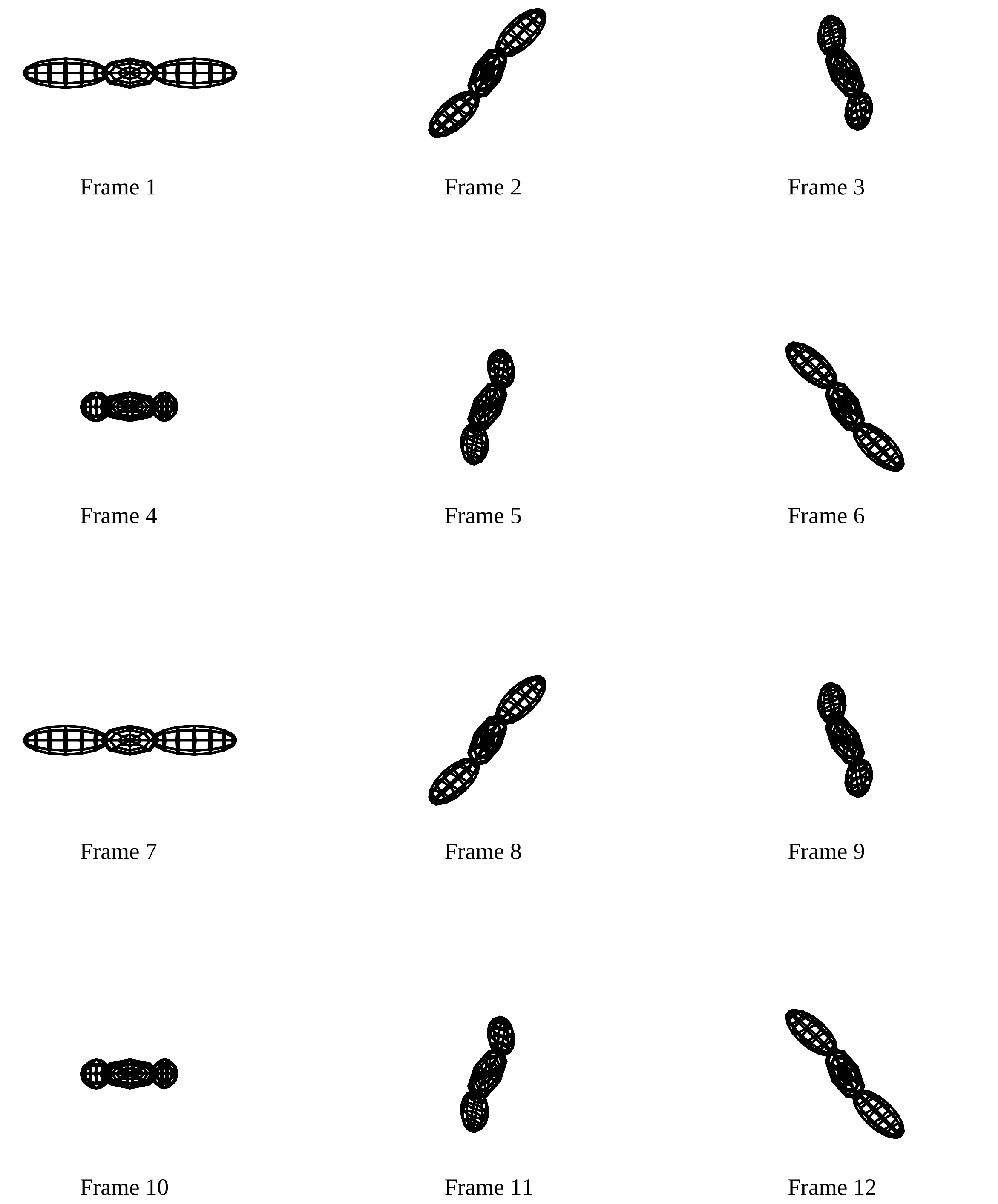

Figure 3.2: Top view of motion created by minimization of acceleration of all points on all bodies. Images 1, 7, and 13 (omitted, but identical to 1) are key frames. □

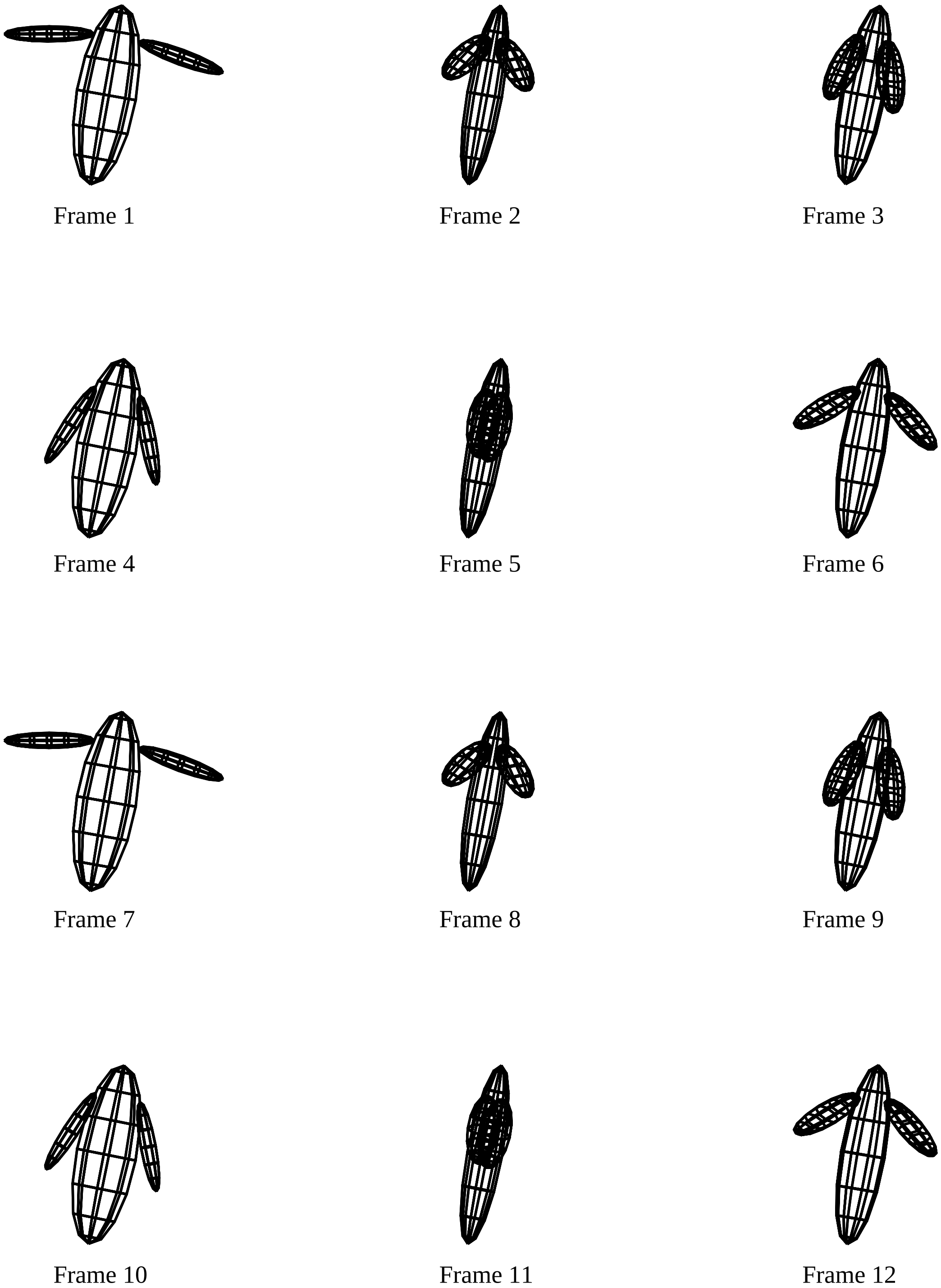

Figure 3.3: Motion created by minimization of covariant acceleration of all points on all bodies. Images 1, 7, and 13 (omitted, but identical to 1) are key frames. □

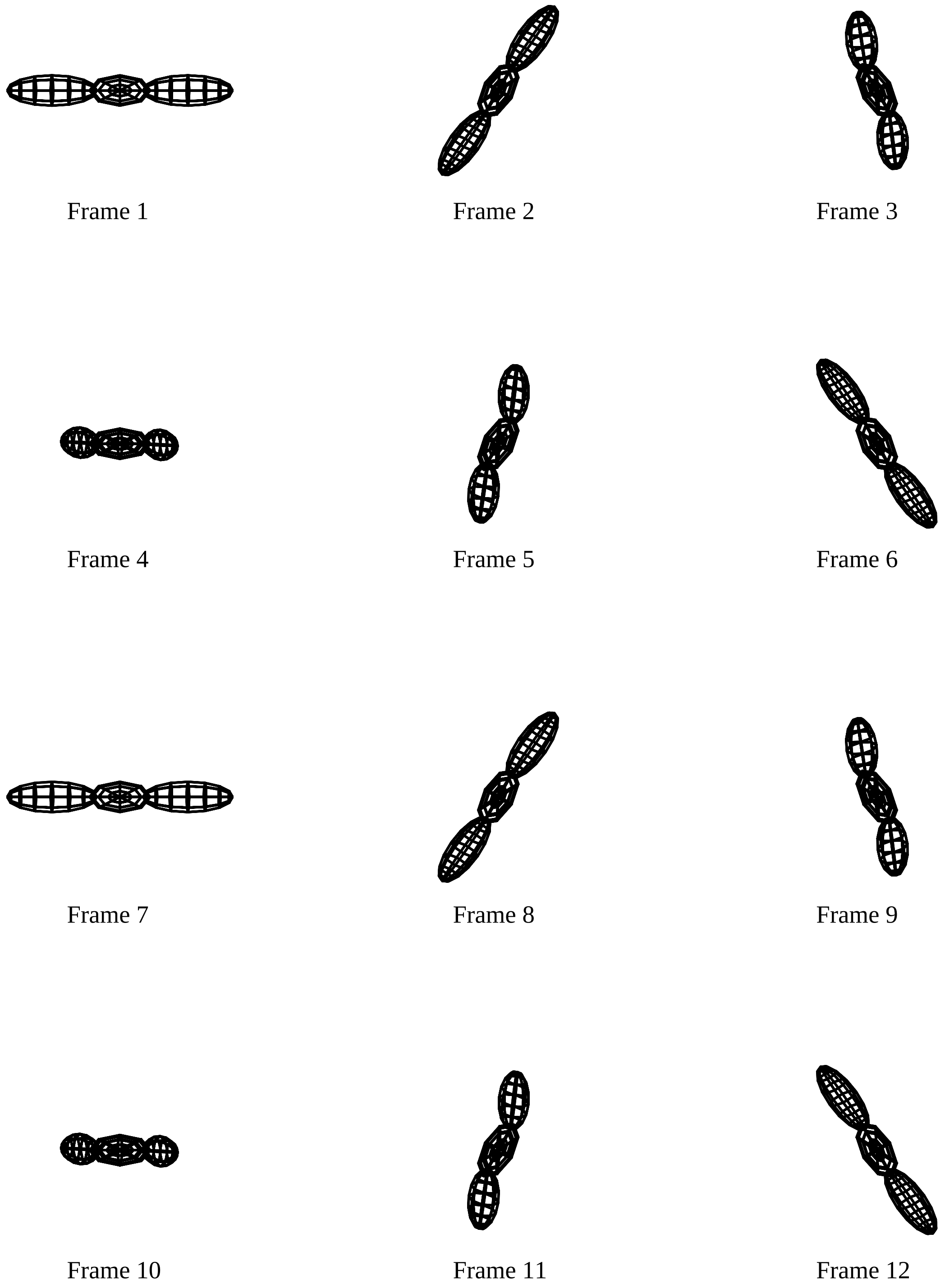

Figure 3.4: Top view of motion created by minimization of covariant acceleration of all points on all bodies. Images 1, 7, and 13 (omitted, but identical to 1) are key frames. □

Appendix A

Notation

A.1 Einstein Summation Notation

Einstein summation notation is a compact way to represent vector and matrix equations. A symbol is given subscripts to represent its rank: one with zero subscripts, x, is a scalar; one with one subscript, x_i, is a vector; one with two subscripts, x_{ij}, is a matrix. Symbols with more than two subscripts are higher-rank objects; for example, one with three subscrips, ϵ_{ijk}, has $n \times n \times n$ components. The length of vectors is sometimes denoted by case of subscripts or other conventions; in this document, we allow an object's size to be determined by context. Usually that size is four unless no quaternions are involved, in which case it is three.

Summations, such as those in dot products and matrix multiplications, need not be expressly written, but are implied by the notation. Repeated subscripts within one term indicate summations; $x_i x_i$ is the normal vector dot product; $M_{ij} x_j$ is the normal matrix-vector multiplication. No subscript may appear more than twice in one term.

A few objects are pre-defined constants. The Kronecker delta symbol, δ_{ij}, is just the identity matrix. It is most often used to change the name of a subscript because

$$\delta_{ij} x_j = x_i. \tag{A.1}$$

The epsilon constant is the "permutation operator." It is defined as:

$$\epsilon_{ijk} = \begin{cases} 0 & \text{if any two of } i, j, \text{ or } k \text{ are equal} \\ 1 & \text{if } (i,j,k) \text{ is an even permutation of } (1,2,3) \\ -1 & \text{if } (i,j,k) \text{ is an odd permutation of } (1,2,3). \end{cases} \tag{A.2}$$

$\epsilon_{ijk} x_j y_k$ is the normal vector cross product of vectors x and y.

The delta and epsilon objects are related by the following identity:

$$\epsilon_{ijk}\epsilon_{ilm} = \delta_{jl}\delta_{km} - \delta_{jm}\delta_{kl}. \tag{A.3}$$

The kappa object, κ_{ijk}, is the quaternion multiplier. Its components are

$$\kappa_{000} = 1 \tag{A.4}$$
$$\kappa_{ij0} = \delta_{ij} \tag{A.5}$$
$$\kappa_{i0k} = \delta_{ik} \tag{A.6}$$
$$\kappa_{0jk} = -\delta_{jk} \tag{A.7}$$
$$\kappa_{ijk} = \epsilon_{ijk}. \tag{A.8}$$

Note that the last equation only holds for non-zero subscript values. $\kappa_{ijk} q^0_j q^1_k$ is the quaternion product of q^0 and q^1.

The quaternion-vector product object is $\vec{\kappa}_{ijk}$. Its components are

$$\vec{\kappa}_{000} = 0 \quad (A.9)$$
$$\vec{\kappa}_{ij0} = 0 \quad (A.10)$$
$$\vec{\kappa}_{i0k} = \delta_{ik} \quad (A.11)$$
$$\vec{\kappa}_{0jk} = -\delta_{jk} \quad (A.12)$$
$$\vec{\kappa}_{ijk} = \epsilon_{ijk}. \quad (A.13)$$

Again, note that the last equation only holds for non-zero subscript values. $\vec{\kappa}_{ijk} q_j x_k$ multiplies a quaternion q by the four-vector $(0, x)$.

The constants and rules described above are only a small part of the ESN lore, but are the symbols and identities used here. For more information, see [Barr].

A.2 Symbols

$\dot{x}$	is the time derivative of x; $\dot{x}$ is equivalent to $\frac{dx}{dt}$
$\frac{D}{dt}$	is the covariant derivative
$\mathcal{I}$	is the moment of inertia tensor, a 3 by 3 matrix
I_{ij}	is the second moment of mass distribution
I	is the moment of inertia scalar, one third of the trace of the 2nd moment of mass distribution Also, $\delta_{ij} I = I_{ij} + \mathcal{I}_{ij}$
∇	is the gradient operator
$\backslash$	is the "without" operator. It projects its first argument into a subspace perpendicular to its second argument.
$\cdot$	is the standard dot product for vectors
$\wedge$	is the standard cross product for vectors
$\times$	is the Cartesian outer product
$\hat{a}$	is an object different from but related to a
u	is normally a dummy variable of integration
q^{-1}	is the quaternion inverse of a quaternion q
B^T	is the matrix transpose of a matrix B
q^0	is quaternion zero
q_i	is quaternion q in ESN
q_0	is the zeroth component of q

Appendix B

Derivations

B.1 Kinetic Energy

The kinetic energy of any body is

$$KE = \frac{1}{2}\int_V \rho(u)v(u,t)\cdot v(u,t)dV. \tag{B.1}$$

Let M^0 be the rotation matrix representing the orientation of the root body and M^1 be similar for the secondary body. Then, for the root body,

$$x(u,t) = M^0(t)u \tag{B.2}$$

so

$$v(u,t) = \dot{x} = (\omega^0 \wedge M^0)u \tag{B.3}$$

or

$$v_i = \epsilon_{ijk}\omega_j^0 M_{kl}^0 u_l \tag{B.4}$$

Substituting into the integral for KE,

$$\begin{aligned}
KE &= \frac{1}{2}\int_V \rho\epsilon_{ijk}\omega_j^0 M_{kl}^0 u_l \epsilon_{iqr}\omega_q^0 M_{rs}^0 u_s dV && \text{(B.5)}\\
&= \frac{1}{2}\int_V \rho(\delta_{jq}\delta_{kr} - \delta_{jr}\delta_{kq})M_{kl}^0 M_{rs}^0 \omega_j^0\omega_q^0 u_l u_s dV && \text{(B.6)}\\
&= \frac{1}{2}\int_V \rho(M_{rl}^0 M_{rs}^0 \omega_q^0\omega_q^0 u_l u_s - M_{ql}^0 M_{js}^0 \omega_j^0\omega_q^0 u_l u_s)dV && \text{(B.7)}\\
&= \frac{1}{2}\int_V \rho(\delta_{ls}\omega_q^0\omega_q^0 u_l u_s - M_{ql}^0 M_{js}^0 \omega_j^0\omega_q^0 u_l u_s)dV && \text{(B.8)}\\
&= \frac{1}{2}\int_V \rho\omega_q^0\omega_q^0 u_s u_s - \rho M_{ql}^0 M_{js}^0 \omega_j^0\omega_q^0 u_l u_s dV && \text{(B.9)}
\end{aligned}$$

Since

$$\begin{aligned}
\int_V \rho u_i u_i dV &= I && \text{(B.10)}\\
\int_V \rho u_i u_j dV &= \delta_{ij} I - \mathcal{I}_{ij} && \text{(B.11)}
\end{aligned}$$

where I is the scalar moment of inertia, and $\mathcal{I}_{ij}$ is the moment of inertia tensor, the integral simplifies to

$$KE = \frac{1}{2}(\omega_q^0\omega_q^0 I - M_{ql}^0 M_{js}^0 \omega_j^0 \omega_q^0(\delta_{ij} I - \mathcal{I}_{ij})) \tag{B.12}$$

$$= \frac{1}{2}\omega_q^0 M_{ql}^0 \mathcal{I}_{ls} M_{js}^0 \omega_j^0. \tag{B.13}$$

For the secondary body, x^1 is similar to a in Figure 2.5, u is the vector from the end of x^1 to each point in the secondary body. M^1 is a rotation matrix corresponding to the orientation of the secondary body, so

$$x(u,t) = M^0 x^1 + M^0 M^1 u \tag{B.14}$$

$$v(u,t) = \dot{x} = (\omega^0 \wedge M^0)x^1 + (\omega^0 \wedge M^0)M^1 u + M^0(\omega^1 \wedge M^1)u \tag{B.15}$$

$$v_i = \epsilon_{ijk}\omega_j^0 M_{kl}^0 x_l^1 + \epsilon_{ijk}\omega_j^0 M_{kl}^0 M_{lm}^1 u_m + M_{ij}^0 \epsilon_{jkl}\omega_k^1 M_{lm}^1 u_m. \tag{B.16}$$

$v_i v_i$ is very long and messy and is the sum of 9 terms, six of which are different. They are (using the same simplifications as for the previous example):

$$1: \quad \omega_p^0\omega_p^0 x_l^1 x_l^1 - \omega_q^0\omega_p^0 M_{pl}^0 M_{qr}^0 x_l^1 x_r^1 \tag{B.17}$$

$$2: \quad \omega_p^0\omega_p^0 M_{ls}^1 x_l^1 u_s - \omega_q^0\omega_p^0 M_{pl}^0 M_{qr}^0 x_l^1 u_s \tag{B.18}$$

$$3: \quad M_{jr}^0 \omega_r^1 \omega_j^0 M_{ls}^1 x_l^1 u_s - M_{jr}^0 M_{rs}^1 \omega_j^0 \omega_l^1 x_l^1 u_s \tag{B.19}$$

$$4: \quad same \quad as \quad 2 \tag{B.20}$$

$$5: \quad \omega_p^0\omega_p^0 u_s u_s - \omega_q^0\omega_k^0 M_{kl}^0 M_{qr}^0 M_{lm}^1 M_{rs}^1 u_m u_s \tag{B.21}$$

$$6: \quad M_{jq}^0 \omega_j^0 \omega_q^1 u_s u_s - M_{jr}^0 M_{qm}^1 M_{rs}^1 \omega_j^0 \omega_q^1 u_m u_s \tag{B.22}$$

$$7: \quad same \quad as \quad 3 \tag{B.23}$$

$$8: \quad same \quad as \quad 6 \tag{B.24}$$

$$9: \quad \omega_q^1\omega_q^1 u_s u_s - \omega_r^1\omega_l^1 M_{lm}^1 M_{rs}^1 u_m u_s \tag{B.25}$$

Integrating $\rho v_i v_i$ is possible because only u is a function of position. It appears in the following forms and integrates to:

$$\int_V \rho dV = m \tag{B.26}$$

$$\int_V \rho u_i dV = x^{cm} \tag{B.27}$$

$$\int_V \rho u_i u_i dV = I \tag{B.28}$$

$$\int_V \rho u_i u_j dV = \delta_{ij} I - \mathcal{I}_{ij} \tag{B.29}$$

Performing the integration and collecting like terms, we get

$$\begin{aligned} KE = {} & \frac{1}{2}(m(\omega_p^0\omega_p^0 x_l^1 x_l^1 - \omega_q^0\omega_p^0 M_{pl}^0 M_{qr}^0 x_l^1 x_r^1) + \\ & 2m x_s^{cm} x_l^1(\omega_p^0\omega_p^0 M_{ls}^1 - \omega_q^0\omega_p^0 M_{pl}^0 M_{qr}^0 M_{rs}^1 + \omega_r^1\omega_j^0 M_{jr}^0 M_{ls}^1 - \omega_l^1\omega_j^0 M_{jr}^0 M_{rs}^1) + \\ & \mathcal{I}_{ms} M_{rs}^1 M_{lm}^1(\omega_q^0\omega_k^0 M_{kl}^0 M_{qr}^0 + 2\omega_l^1\omega_j^0 M_{jr}^0 + \omega_r^1\omega_l^1)) \end{aligned} \tag{B.30}$$

B.2 Hermite Polynomial

Given:

an interval, $t \in [0, 1]$

q^0, a quaternion representing a rotation

q^1, a quaternion representing a rotation

ω^0, a vector $(0, \omega)$ that includes an angular velocity

ω^1, similar to ω^0

we want to find the coefficients of the cubic polynomial

$$S(t) = a\ t^3 + b\ t^2 + c\ t + d \tag{B.31}$$

such that

$S(0) = q^0$

$S(1) = q^1$

$\dot{S}(0) = \frac{1}{2}\omega^0 q^0$

$\dot{S}(1) = \frac{1}{2}\omega^1 q^1$

Taking the time derivative of $S(t)$, we get

$$\dot{S}(t) = 3a\ t^2 + 2b\ t + c \tag{B.32}$$

Because we are given

$$S(0) = q^0 \tag{B.33}$$

and we know by substitution

$$S(0) = d \tag{B.34}$$

we see that

$$d = q^0. \tag{B.35}$$

Substituting zero and one into the equations for $S(t)$ and $\dot{S}(t)$, we get four equations in four unknowns:

$$\begin{aligned} q^0 &= d & \text{(B.36)}\\ q^1 &= a + b + c + d & \text{(B.37)}\\ \frac{1}{2}\omega^0 q^0 &= c & \text{(B.38)}\\ \frac{1}{2}\omega^1 q^1 &= 3a + 2b + c. & \text{(B.39)} \end{aligned}$$

Solving this system, we get

$$\begin{aligned} a &= 2q^0 - 2q^1 + \frac{1}{2}\omega^0 q^0 - \frac{1}{2}\omega^1 q^1 & \text{(B.40)}\\ b &= -3q^0 + 3q^1 - \omega^0 q^0 + \frac{1}{2}\omega^1 q^1 & \text{(B.41)}\\ c &= \frac{1}{2}\omega^0 q^0 & \text{(B.42)}\\ d &= q^0. & \text{(B.43)} \end{aligned}$$

$S(t)$ can be represented, therefore, as a matrix spline:

$$S(t) = (t^3 \ t^2 \ t \ 1) \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} q^0 \\ q^1 \\ \frac{1}{2}\omega^0 q^0 \\ \frac{1}{2}\omega^1 q^1 \end{pmatrix}. \tag{B.44}$$

This is a Hermite polynomial and is usually written as

$$S(t) = T \ H \ K \tag{B.45}$$

with factors corresponding to the ones in the previous equation.

Appendix C

Quaternion Arithmetic

Quaternions are extended complex numbers with bases being 1 and three square roots of -1, usually denoted $\mathbf{i}, \mathbf{j}$, and $\mathbf{k}$. A quaternion $q = (q_0, q_1, q_2, q_3)$ can also be written $q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$. Multiplication of $\mathbf{i}, \mathbf{j}$, and $\mathbf{k}$ is not commutative; $\mathbf{ij} = \mathbf{k}$, but $\mathbf{ji} = -\mathbf{k}$.

Quaternion addition is component-by-component and is not a common operation.

Quaternion multiplication, $q = pr$, can be computed by

$$\begin{aligned} q_0 &= p_0r_0 - p_1r_1 - p_2r_2 - p_3r_3 && \text{(C.1)} \\ q_1 &= p_0r_1 + p_1r_0 + p_2r_3 - p_3r_2 && \text{(C.2)} \\ q_2 &= p_0r_2 - p_1r_3 + p_2r_0 + p_3r_1 && \text{(C.3)} \\ q_3 &= p_0r_3 + p_1r_2 - p_2r_1 + p_3r_0. && \text{(C.4)} \end{aligned}$$

Quaternion multiplication is also associative, but not commutative.

Quaternion inversion is

$$q^{-1} = (q_0, -q_1, -q_2, -q_3)/(q \cdot q). \tag{C.5}$$

Quaternions can be used to represent rotations. Only the direction of the quaternion is used; the magnitude is assumed to be unitary. The last three components of a quaternion are the axis around which the rotation occurs; the first component represents the magnitude of the rotation, θ by

$$\theta = 2cos^{-1}(q_0) \tag{C.6}$$

In order to perform the rotation, a vector is augmented by the addition of a zero zeroth component, and the composite form, $\hat{v} = (0, v)$, is premultiplied by the quaternion representing the desired rotation, and postmultiplied by that quaternion's inverse. That is,

$$\hat{v}_{rotated} = q\ \hat{v}\ q^{-1} \tag{C.7}$$

Rotation by quaternion is associative. That is,

$$(pq)\hat{v}(pq)^{-1} = p(q\ \hat{v}\ q^{-1})p^{-1} \tag{C.8}$$

For much more about quaternions, see [Shoemake] or [Shoemake 89].

Bibliography

[Bartels] Bartels, R. H., Beatty, J. C., and Barsky, B. A., *An Introduction to the Use of Splines in Computer Graphics,* Morgan Kaufmann, 1987.

[Barr] Barr, A. H., "The Einstein Summation Notation," Technical Report, Computer Graphics Group, California Institute of Technology, Pasadena, California.

[Barr 81] Barr, A. H., "Superquadrics and Angle-Preserving Transformations," IEEE Computer Graphics and Applications, Vol. 1. No. 1, pp. 11-23, January, 1981.

[Barr 92] Barr, A. H., Currin, B., Gabriel, S., and Hughes, J. F., "Smooth Interpolation of Orientations with Angular Velocity Constraints Using Quaternions," SIGGRAPH '92 Conference Proceedings, pp. 313-320, July, 1992.

[Barzel] Barzel, R., *Physically-Based Modeling for Computer Graphics: A Structured Approach*, Academic Press, London, 1992.

[Do Carmo] Do Carmo, M. P., *Differential Geometry of Curves and Surfaces*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[Gabriel] Gabriel, S., and Kajiya, J., "Spline Interpolation in Curved Space," in course notes, "State of the Art in Image Synthesis," SIGGRAPH, 1985.

[Gill] Gill, P. E., Murray, W., and Wright, M. H., *Practical Optimization*, Academic Press, London, 1981.

[Lamport] Lamport, L., *LaTeX, A Document Preparation System*, Addison-Wesley, Reading, Massachusetts, 1986.

[Minos] Murtagh, B. A., and Saunders, M. A., *MINOS 5.1 User's Guide,* Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, December 1983.

[Misner] Misner, C. W., Thorne, K. S., and Wheeler, J. A., *Gravitation*, Freeman, San Francisco, 1973.

[NAG] Numerical Applications Group, NAG Fortran Library, NAG Ltd, Oxford, UK, April 1990.

[Press] Press *et al. Numerical Recipes*, Cambridge University Press, Cambridge, 1986.

[Shoemake] Shoemake, K., "Animating Rotation with Quaternion Curves," SIGGRAPH '85 Conference Proceedings, pp. 245-254, July, 1985.

[Shoemake 89] Shoemake, K., "Quaternion Calculus for Animation," in course notes "Math for SIGGRAPH," SIGGRAPH, 1989.

[Witkin] Witkin, A., and Kass, M., "Spacetime Constraints", SIGGRAPH '88 Conference Proceedings, pp. 159-168, August, 1988.