Rigid Body Simulation with Contact and Constraints

by

Michael Bradley Cline

B.S., The University of Texas at Austin, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

July 2002

© Michael Bradley Cline, 2002

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMPUTER SCIENCE

The University of British Columbia
Vancouver, Canada

Date JULY 18th 2002

Abstract

We present techniques for constructing an interactive rigid body simulation system, and describe our implementation such a system. We use a constraint-based simulation approach, incorporating a time stepping method which handles multiple contacts with friction in a general way.

We describe a method for constraint stabilization for simulations with contact. The method is based on the post-stabilization methods of Ascher and Chin, but is formulated as a linear complementarity problem, allowing us to solve for the stabilization step when there are redundant inequality (contact) constraints.

As an application of our simulation software, we present a method for human character animation where motion capture data is used to drive a rigid body simulation. This allows physical constraints and forces acting on the character to be taken into account.

Contents

List of Figures

Acknowledgements

I am very grateful to my supervisor, Dinesh Pai. Without Dinesh's guidance, this thesis would not have been possible. Special thanks also go to Uri Ascher for agreeing to be my second reader, and providing me with many helpful comments. KangKang Yin deserves a great deal of credit for the human simulation examples in Chapter 5, and for patiently helping to weed out the numerous bugs in the simulation code. I thank Paul Kry for many interesting discussions in the lab, and for getting me outside for some much needed unicycling breaks during the summer months when I was hard at work on this thesis. On a personal level, I would like to thank Christy for being there through the good times and the hard times of the last three years, providing motivation and emotional support. My parents, Bob and Ineke, also deserve thanks for their support.

MICHAEL BRADLEY CLINE

The University of British Columbia
July 2002

Chapter 1

Introduction

Predicting the motion of real-world objects using computers has a history almost as old as computers themselves. Some of the very first electronic computers were designed in secrecy, at great expense, with the sole purpose of calculating tables to predict the motion of the U.S. Navy's projectile weapons. The past couple of decades have seen an explosion in amount of research on physics-based animation, especially rigid body motion, by robotics and computer graphics researchers. Applications of this technology range from virtual reality and games to simulating and planning robot motion.

This thesis makes two main contributions to the area of rigid body simulation. The first contribution is a description of a constraint stabilization method specifically designed to handle both contact constraints and equality constraints. Our stabilization is based on the post-stabilization methods described by Ascher, Chin, et al. [Asc97, ACR94], but is phrased as a linear complementarity problem to allow stabilization of contact constraints. This fits nicely with our dynamics framework, which is based on the time-stepping schemes of Anitescu et al. [AP97]. We describe our stabilization method in Chapter 3.

The second contribution of this thesis is a detailed description of the architecture of our rigid body simulation software. Although there is a large amount of published work relating to rigid body physics, implementing a general purpose rigid body simulation system remains a challenging task. Most papers related to the subject describe only a small component, and figuring out how all of the pieces fit together is non-trivial. Our system architecture is explained in Chapter 4.

We present an application of our simulation software in Chapter 5 where we use some novel techniques for human animation. We show how to extend and use the simulator for dynamic simulation of human motions from motion capture data. The simulated humans are driven using motor controls which are based on concepts from biomechanics and neuroscience research.

1.1 Related Work

Dynamic simulation of rigid bodies has a long history, going back over two decades. We cannot hope to list all of the work that has been published on this topic. We mention here the work that was most influential to this thesis.

Some of the first published work in this area was on simulating the dynamics of articulated rigid bodies (sets of rigid bodies connected by joints), using a reduced coordinate approach. Vereshchagin [Ver74], in 1974, and Armstrong and Green [AG85], published efficient algorithms for articulated body dynamics. In the late eighties, Featherstone's [Fea87] book laid down a mathematical foundation for rigid multibody simulation.

Papers by Barzel and Barr [BB88] and Witkin et al. [WGW90] were earlier works that introduced constraint-based systems using Lagrange multipliers. These systems used full coordinates for each rigid body, and implemented constraints

through *constraint forces*. However, these works dealt primarily with bilateral constraints, mostly ignoring the issues of contact. Originally, these constraint-based methods also suffered from slow computation algorithms. For instance, [BB88] reported using singular value decomposition to solve the constraint force equations, a technique which was relatively slow and did not take advantage of sparsity in the constrained motion equations. Lubich [LNPE92] later demonstrated techniques to solve such systems in linear time using sparse matrix techniques.

Around the same time as [BB88], Moore and Wilhelms [MW88], and Hahn [Hah88] published methods for simulating contact. Neither addressed contact in the general case, where there can be multiple bodies in contact, and multiple contacts on the same body. Modelling contact, and in a general and robust way, has remained a difficult problem. Mirtich and Canny [MC95] continued work along the lines of [MW88] and [Hah88], using impulse based methods which were well suited to temporary contacts and single contact points, but would have difficulty simulating, for instance, large stacks of objects statically resting on each other.

Many others have worked on extending the constraint-based methods to handle contact. This is typically done by formulating the constrained dynamics equations as a linear complementarity problem (LCP), which is essentially a problem of finding some solution to a linear system that satisfies certain inequality constraints. The first paper to pose the contact problem as an LCP was published by Lötstedt [L82] in 1982. Baraff later presented a method [Bar94] that used an LCP algorithm by Cottle and Dantzig [CD68] to solve contact and static friction forces at interactive rates.

One of the difficulties with earlier LCP methods was that there is no guarantee of the existence of a solution, in the presence of contact with Coulomb friction.

Indeed, there are known configurations of rigid bodies for which no set of contact and friction forces satisfies all of the constraints (such as the Painlevé paradox). Using a model that allows impulsive forces turned out to be the key to avoiding these problems. Stewart and Trinkle [ST96] introduced a time-stepping scheme that combines the acceleration-level LCP with a time step, to obtain an LCP where the variables are velocities and impulses. Impulsive forces are incorporated naturally in such a model. Their method guaranteed solvability for a larger class of problems, and was later modified by Anitescu and Potra [AP97], leading to an algorithm that guaranteed solvability regardless of the configuration or number of contacts. Our work is based heavily on [AP97].

The stabilization method that we will present is based on work by Ascher and Chin [Asc97] [ACR94]. These papers discuss stabilization methods in general and post-stabilization in particular as the favoured method. We will draw on this work in Chapter 3, and compare post-stabilization and Baumgarte stabilization [Bau72] for counteracting constraint drift in our rigid body simulations.

1.2 Thesis Organization

The layout of this thesis is as follows:

The next chapter will present background material on rigid body physics laws and simulation algorithms. We will begin by explaining the various ways of parameterizing the state of a rigid body (the position, orientation, and velocity). We will give the equations of motion. We will explain how to mathematically represent constrained motion, and then discuss how to solve the motion equations with constraints. Contact constraints will be given special attention, and we will describe how the problem of dynamics with contact is modelled as a linear complementarity

problem (LCP). This is followed by a description of Lemke's algorithm, which is the method we use to solve the LCP. We finish the chapter by presenting the time-stepping method given by Anitescu and Potra [AP97], which our implementation is based on.

In Chapter 3, we explain the need for constraint stabilization in a system like ours. We will present Baumgarte's method as one example of a stabilization technique, then we will present our version of the Ascher-Chin post-stabilization, and explain how and why we use an LCP similar to the dynamics LCP to accomplish the post-stabilization.

Chapter 4 gives the details of our system's architecture. We explain how we have implemented the methods discussed in the previous chapters.

Chapter 5 presents the results of our work. Two of the goals of our simulation software were to be extensible, and to run at interactive rates. To demonstrate that these goals were met, we show an application of our software to human character animation. By combining motion capture data with a rigid body simulation, we are able to generate perturbed animations of characters reacting to external physical constraints. We can do this at interactive frame rates ($\geq$30 fps) on a modern personal computer. In this chapter we also present some results from experiments that test our post-stabilization method.

In Chapter 6, we will conclude the thesis and discuss opportunities for future work.

Chapter 2

Background

2.1 Position and Rotation of a Rigid Body

An unconstrained rigid body has six degrees of freedom: three degrees of translational freedom, and three degrees rotational freedom. There are many possible ways to parameterize the position and rotation of a body.

The translational position is usually given as a vector from the origin to some fixed point on the body. The centre of mass of the body is a convenient choice for this fixed point, but any point can be used.

Parameterizing rotation of a rigid body in 3D is more tricky. There are several methods commonly used, each with its own advantages and disadvantages.

2.1.1 Rotation Matrices

Within computer graphics, rotation matrices are the most well known method of representing rotation. Any rotation in 3D can be parameterized by a 3×3 orthogonal matrix. Rotation matrices are convenient because in addition to parameterizing the rotation, they can also be used to transform vectors from one coordinate frame

to another.

Together, the rotation matrix $\mathbf{R}$ and position vector $\boldsymbol{p}$ define a rigid transformation $\boldsymbol{g}$. Using homogeneous coordinates, we write this transformation as a 4×4 matrix

$$\boldsymbol{g} = \begin{pmatrix} \mathbf{R} & \boldsymbol{p} \\ 0 & 1 \end{pmatrix}. \tag{2.1}$$

A drawback of using rotation matrices is that they use nine numbers to parameterize three degrees of freedom. In terms of memory requirements, this means that rotation matrices take up more space than they really need to. Another difficulty is that in the presence of numerical error, it is possible to arrive at a non-orthogonal matrix. Thus it is sometimes necessary to do a costly re-normalization to ensure that the matrix actually corresponds to a valid rotation.

2.1.2 Euler Angles / Roll, Pitch, Yaw

According to Euler's rotation theorem (see [BR79]), any rotation can be described by three angles. Thus any rotation matrix $\mathbf{R}$ can be parameterized by three angles (α, β, γ):

$$\mathbf{R} = rot(\boldsymbol{v}_1, \alpha) \times rot(\boldsymbol{v}_2, \beta) \times rot(\boldsymbol{v}_3, \gamma), \tag{2.2}$$

where $\boldsymbol{v}_1, \boldsymbol{v}_2$ and $\boldsymbol{v}_3$ are a specific set of axes, and $rot(\boldsymbol{v}, \theta)$ is the rotation matrix corresponding to a rotation of angle θ about the axis $\boldsymbol{v}$. There are many different conventions dictating which axes are actually used, and in which order. The most common convention may be the ZXZ convention, which means rotations are parameterized by a rotation about the Z-axis, followed by the X-axis, and then

about the Z-axis again. Another common convention is the ZYX convention, which is also known as *roll, pitch and yaw*[1].

Although the Euler angle representation avoids the problems of over-parameterization that rotation matrices suffer from, they have problems of their own. All 3-angle parameterizations suffer from singularities where two of the rotation axes line up and a single rotation can be parameterized by an infinite number of 3-angle combinations. For example, using the ZXZ convention, if the second angle is zero, then the first and third axes line up. Thus a rotation of θ about the Z-axis can be parameterized by any triplet $(\alpha, 0, \gamma)$ such that $\alpha + \gamma = \theta$. Singularities are undesirable because near the singularity small changes in position can result in extremely large changes in the Euler angles, leading to numerical problems.

Another drawback of using Euler angles is that they cannot be used to transform vectors directly from one frame to another the way rotation matrices can, nor can rotations be easily combined. The Euler angles have to be converted to matrices first, as in Equation 2.2.

2.1.3 Unit Quaternions

Shoemake [Sho85] introduced unit length quaternions to the computer graphics community as a means of representing rotation. Quaternions are often preferable to using rotation matrices or Euler angles. They avoid the main problems of the other methods, and offer some additional benefits such as providing an easy way to correctly interpolate between rotations.

[1] In the engineering and robotics communities, the term *Euler Angles* is usually taken to mean ZXZ, and other terminology (e.g., Fick angles, Helmholtz angles, roll-pitch-yaw) are used to name the other conventions[MLS94]. Computer graphics papers tend to confuse the matter by either defining *Euler Angles* to mean XYZ angles [Mat, GOM98], or by using the term *Euler Angles* more loosely, to describe any 3-angle parameterization.

Quaternions are a generalization of complex numbers. However, their relationship to complex numbers is not of much importance to this thesis, and it is more convenient to think of a quaternion as a pair $\boldsymbol{q} = (\boldsymbol{v}, s)$ containing a 3-vector and a scalar (sometimes we will treat it as a 4-vector). A rotation of angle θ about the axis $\boldsymbol{v}$ (a unit vector) is given as the quaternion

$$\begin{pmatrix} \sin(\theta/2) \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \\ \cos(\theta/2) \end{pmatrix}. \tag{2.3}$$

Like rotation matrices, quaternions must sometimes be re-normalized to make sure they correspond to valid rotations. However, the cost of normalizing a quaternion is much less than the cost of making a 3×3 matrix orthogonal.

Quaternions do not suffer from the singularity problems of Euler Angles. The mapping between quaternions and rotations is two-to-one because every quaternion represents the same rotation as its negative.

Rotations can be easily combined by multiplying two quaternions together. The rule for multiplying quaternions together is

$$\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 = \begin{pmatrix} \boldsymbol{v}_1 \\ s_1 \end{pmatrix} \otimes \begin{pmatrix} \boldsymbol{v}_2 \\ s_2 \end{pmatrix} = \begin{pmatrix} s_1\boldsymbol{v}_2 + s_2\boldsymbol{v}_1 + \boldsymbol{v}_1 \times \boldsymbol{v}_2 \\ s_1 s_2 - \boldsymbol{v}_1 \cdot \boldsymbol{v}_2 \end{pmatrix}. \tag{2.4}$$

The inverse of a quaternion is also easy to find, simply by negating either the vector or scalar part of the quaternion.

$$\boldsymbol{q}^{-1} = \begin{pmatrix} q_x \\ q_y \\ q_z \\ q_w \end{pmatrix}^{-1} = \begin{pmatrix} -q_x \\ -q_y \\ -q_z \\ q_w \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} q_x \\ q_y \\ q_z \\ -q_w \end{pmatrix}. \tag{2.5}$$

Quaternions provide an easy way to transform a vector between coordinate frames. If rotation matrix $\mathbf{R}$ and quaternion $\boldsymbol{q}$ both represent the same rotation, then

$$\mathbf{R}\boldsymbol{v} = unquat(\boldsymbol{q} \otimes quat(\boldsymbol{v}) \otimes \boldsymbol{q}^{-1}). \tag{2.6}$$

It the equation above, the *quat* function constructs a quaternion from a vector by appending a zero as the scalar part, and the *unquat* returns the vector part of the quaternion.

In the rest of this thesis, when we talk about the *position* of a rigid body, we are referring implicitly to both the position and orientation of the body. We will concatenate the position vector $\boldsymbol{p}_i$ and the quaternion $\boldsymbol{q}_i$ into a single vector $\mathbf{p}_i = \begin{pmatrix} \boldsymbol{p}_i \\ \boldsymbol{q}_i \end{pmatrix}$. Often we concatenate the positions of many rigid bodies into a single vector $\mathbf{p} = \begin{pmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_n \end{pmatrix}$.

2.2 Velocity of a Rigid Body

Rigid body velocity is usually represented by a combination of an angular velocity vector, $\boldsymbol{\omega}$, and a linear velocity vector $\boldsymbol{v}$. The angular velocity vector's direction is the axis of rotation, and the magnitude is the rate at which the body is spinning (radians per unit time). The linear velocity vector corresponds to the velocity of some fixed point on the rigid body. As with representing the position of a rigid body, there is a choice of which point is used. There are two different choices that are commonly used. In the graphics literature, $\boldsymbol{v}$ is usually the velocity of the body's centre of mass (e.g., see [Hah88], [Bar97]). This is convenient because the centre of mass has the same equations of motion as a particle. The other option (which is common in the robotics literature) is to let $\boldsymbol{v}$ stand for the velocity of the point attached to the rigid body that is at the origin of the coordinate frame.

In Section 2.4, we will discuss *twist coordinates*, which are used in the robotics literature to describe rigid body velocity.

For any vector $\boldsymbol{x}$ that is fixed to a rigid body with angular velocity $\boldsymbol{\omega}$, the rate of change of $\boldsymbol{x}$ is given by $\boldsymbol{\omega} \times \boldsymbol{x}$. Because we would like to use this fact later in matrix equations, it is useful to define an operator that gives the *cross-product matrix* of a vector. We will put brackets around a vector to denote that vector's cross product matrix:

$$\boldsymbol{\omega} \times \boldsymbol{x} = [\boldsymbol{\omega}]\boldsymbol{x} = \begin{pmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{pmatrix} \boldsymbol{x}. \tag{2.7}$$

2.2.1 The Velocity of a Point on a Rigid Body

Given the linear velocity $\boldsymbol{v}$ and angular velocity $\boldsymbol{\omega}$ of a rigid body, we can determine the velocity of a point attached to the body. Assume that $\boldsymbol{v}$ is the velocity of a point $\boldsymbol{p}_0$ attached to the body, then any other point $\boldsymbol{p}$ has the velocity

$$\dot{\boldsymbol{p}} = \boldsymbol{v} + [\boldsymbol{\omega}](\boldsymbol{p} - \boldsymbol{p}_0). \tag{2.8}$$

2.2.2 The Relationship Between Angular Velocity and Quaternions

If $\boldsymbol{\omega}$ is the angular velocity vector for a rigid body, and $\boldsymbol{q}$ is the quaternion representing that body's orientation, then the rate of change of $\boldsymbol{q}$ is given by

$$\dot{\boldsymbol{q}} = \frac{1}{2}\, quat(\boldsymbol{\omega}) \otimes \boldsymbol{q}. \tag{2.9}$$

Rearranging this equation, we can determine the angular velocity from $\dot{\boldsymbol{q}}$.

$$\begin{aligned} 2\dot{\boldsymbol{q}} &= quat(\boldsymbol{\omega}) \otimes \boldsymbol{q}, \\ 2\dot{\boldsymbol{q}} \otimes \boldsymbol{q}^{-1} &= quat(\boldsymbol{\omega}) \otimes \boldsymbol{q} \otimes \boldsymbol{q}^{-1}, \\ 2\dot{\boldsymbol{q}} \otimes \boldsymbol{q}^{-1} &= quat(\boldsymbol{\omega}), \\ \boldsymbol{\omega} &= unQuat(2\dot{\boldsymbol{q}} \otimes \boldsymbol{q}^{-1}). \end{aligned} \tag{2.10}$$

2.3 Equations of Motion

The equation of motion for a rigid body can be derived by thinking of the body as a collection of particles, then integrating over the entire mass [Mas01]. The first result of this calculation is a proof that the centre of mass of any rigid body behaves like a particle. Thus, if $\boldsymbol{v}$ is the velocity of the centre of mass, $\boldsymbol{f}$ is the sum of all

forces acting on the body, and m is the total mass of the body, the acceleration obeys Newton's second law:

$$\boldsymbol{f} = m\frac{d\boldsymbol{v}}{dt}. \tag{2.11}$$

Another way of saying the same thing is to say that *force is the rate of change of linear momentum* ($\boldsymbol{f} = \frac{d}{dt}(m\boldsymbol{v})$).

Whereas linear momentum is related to linear velocity with a scalar (the mass), angular momentum is related to angular velocity with a matrix $\mathcal{I}$, called the *angular inertia matrix.* The reason for this is that objects generally have different angular inertias around different axes of rotation. Angular momentum is defined as $\mathcal{I}\boldsymbol{\omega}$. The total torque $\boldsymbol{\tau}$ applied to the body is equal to the rate of change of the angular momentum:

$$\boldsymbol{\tau} = \frac{d}{dt}(\mathcal{I}\boldsymbol{\omega}). \tag{2.12}$$

When the body's distribution is approximated by a collection of k particles, the angular inertia matrix is given by

$$\mathcal{I} = \sum_{i=1}^{k} m_i(\boldsymbol{r}_i^T\boldsymbol{r}_i\mathbf{I} - \boldsymbol{r}_i\boldsymbol{r}_i^T), \tag{2.13}$$

where m_i is the mass of the i'th particle, and $\boldsymbol{r}_i$ is the vector from the origin to the i'th particle. $\mathbf{I}$ here is the 3×3 identity matrix. If $\boldsymbol{r}_i = (x_i, y_i, z_i)$ then

$$\mathcal{I} = \sum_{i=0}^{k} m_i \begin{pmatrix} y_i^2 + z_i^2 & -x_iy_i & -x_iz_i \\ -x_iy_i & x_i^2 + z_i^2 & -y_iz_i \\ -x_iz_i & -y_iz_i & x_i^2 + y_i^2 \end{pmatrix}. \tag{2.14}$$

In body coordinates, the angular inertia matrix $\mathcal{I}$ is constant. The world coordinates of the angular inertia matrix are given by $\mathbf{R}\mathcal{I}\mathbf{R}^T$, where $\mathbf{R}$ is the rotation matrix of the body. So, we can evaluate Equation 2.12 as follows:

$$\begin{aligned}
\boldsymbol{\tau} &= \frac{d}{dt}(\mathcal{I}\boldsymbol{\omega}) \\
&= \mathcal{I}\dot{\boldsymbol{\omega}} + \dot{\mathcal{I}}\boldsymbol{\omega} \\
&= \mathcal{I}\dot{\boldsymbol{\omega}} + \frac{d}{dt}(\mathbf{R}\mathcal{I}_{body}\mathbf{R}^T)\boldsymbol{\omega} \\
&= \mathcal{I}\dot{\boldsymbol{\omega}} + (\dot{\mathbf{R}}\mathcal{I}_{body}\mathbf{R}^T + \mathbf{R}\mathcal{I}_{body}\dot{\mathbf{R}}^T)\boldsymbol{\omega} \\
&= \mathcal{I}\dot{\boldsymbol{\omega}} + ([\boldsymbol{\omega}]\mathbf{R}\mathcal{I}_{body}\mathbf{R}^T + \mathbf{R}\mathcal{I}_{body}\mathbf{R}^T\hat{\boldsymbol{\omega}})\boldsymbol{\omega} \\
&= \mathcal{I}\dot{\boldsymbol{\omega}} + [\boldsymbol{\omega}]\mathcal{I}\boldsymbol{\omega} - \mathcal{I}[\boldsymbol{\omega}]\boldsymbol{\omega}.
\end{aligned} \tag{2.15}$$

The final term cancels out since $\boldsymbol{\omega} \times \boldsymbol{\omega}$ is zero. This leaves us with a relationship known as *Euler's Equation*:

$$\boldsymbol{\tau} = \mathcal{I}\dot{\boldsymbol{\omega}} + [\boldsymbol{\omega}]\mathcal{I}\boldsymbol{\omega}. \tag{2.16}$$

2.4 Twist/Wrench/Adjoint Notation

In the robotics literature we find a more general notation and language for describing rigid body velocities and forces that act on rigid bodies. We will now introduce vectors called *twists*, which describe velocities, and *wrenches*, which describe forces, and explain how these objects transform from one coordinate frame to another. Other sources [MLS94] derive these quantities from exponential equations of rotation. Here we will concentrate instead on the relationship of these quantities to the "centre-of-mass" parameterizations mentioned earlier.

2.4.1 Twists

A twist is a vector that expresses rigid motion or velocity. In Section 2.2, we saw how to parameterize the velocity of a rigid body as a linear velocity vector and an angular velocity vector. The coordinates of a twist are given as a 6-vector $\mathbf{v} = \begin{pmatrix} \omega \\ v \end{pmatrix}$ containing a linear velocity vector $\boldsymbol{v}$ and an angular velocity vector $\boldsymbol{\omega}$. According to the way twists are defined, the velocity vector $\boldsymbol{v}$ is not the velocity of the body's centre of mass, but the velocity of a point attached to the body, located at the origin of the coordinate frame.

Since our implementation assumes that the velocity vector $\boldsymbol{v}$ is the velocity of the body's centre of mass, and many computer graphics papers give their equations in terms of the motion of the centre of mass, we feel it is important to discuss how our parameterization relates to twists. If $\boldsymbol{v}$ is the velocity of the body's centre of mass, given in world coordinates, and $\boldsymbol{\omega}$ is the angular velocity in world coordinates, then $\begin{pmatrix} \omega \\ v \end{pmatrix}$ can be seen as a twist in a coordinate frame with its origin at the centre of mass, but aligned with the axes of the world coordinate frame.

Earlier, we gave an equation for the velocity of a point $\boldsymbol{p}$ attached to a rigid body as

$$\dot{\boldsymbol{p}} = \boldsymbol{v} + [\boldsymbol{\omega}](\boldsymbol{p} - \boldsymbol{p}_0). \tag{2.17}$$

If $\begin{pmatrix} \omega \\ v \end{pmatrix}$ is a twist, then $\boldsymbol{p}_0$ above is the origin, and Equation 2.17 simplifies to

$$\dot{\boldsymbol{p}} = \boldsymbol{v} + [\boldsymbol{\omega}]\boldsymbol{p}. \tag{2.18}$$

If we use homogeneous coordinates for $\boldsymbol{p}$, we can write Equation 2.18 as

$$\dot{\boldsymbol{p}} = \begin{pmatrix} [\boldsymbol{\omega}] & \boldsymbol{v} \\ 0 & 0 \end{pmatrix} \boldsymbol{p}. \tag{2.19}$$

Accordingly, we define a new bracket operator that acts on a twist $\mathbf{v} = \begin{pmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{pmatrix}$,

$$[\mathbf{v}] = \begin{pmatrix} [\boldsymbol{\omega}] & \boldsymbol{v} \\ 0 & 0 \end{pmatrix}, \tag{2.20}$$

so that Equation 2.19 becomes

$$\dot{\boldsymbol{p}} = [\mathbf{v}]\boldsymbol{p}. \tag{2.21}$$

2.4.2 Wrenches

A wrench is a vector that expresses force and torque acting on a body. A wrench $\mathbf{f} = \begin{pmatrix} \boldsymbol{\tau} \\ \boldsymbol{f} \end{pmatrix}$ contains an angular component $\boldsymbol{\tau}$ and a linear component $\boldsymbol{f}$, which are applied at the origin of the coordinate frame they are specified in.

2.4.3 Transforming Twists and Wrenches Between Coordinate Frames

We can transform twists and wrenches between coordinate frames by multiplying them with a matrix called the *adjoint transformation.* If the 4×4 matrix that transforms points and vectors from frame A to frame B is

$${}^{b}_{a}\mathbf{E} = \begin{pmatrix} \mathbf{R} & \boldsymbol{p} \\ 0 & 1 \end{pmatrix}, \tag{2.22}$$

then the 6×6 adjoint that transforms twists from frame A to frame B is

$$ {}^{b}_{a}\mathbf{Ad} = \begin{pmatrix} \mathbf{R} & 0 \\ [p]\mathbf{R} & \mathbf{R} \end{pmatrix}. \tag{2.23} $$

Throughout this thesis, twists will often have a leading superscript to denote their frame. Twists transform from frame to frame according to the following rule:

$$ {}^{b}\mathbf{v} = {}^{b}_{a}\mathbf{Ad}\ {}^{a}\mathbf{v}. \tag{2.24} $$

The inverse of ${}^{b}_{a}\mathbf{Ad}$ is ${}^{a}_{b}\mathbf{Ad}$, which transforms a twist from frame B to frame A:

$$ {}^{a}\mathbf{v} = {}^{b}_{a}\mathbf{Ad}^{-1}\ {}^{b}\mathbf{v} = {}^{a}_{b}\mathbf{Ad}\ {}^{b}\mathbf{v}. \tag{2.25} $$

Wrenches transform differently than twists, and will be written with a leading subscript. The rule for transform a wrench $\mathbf{f}$ from frame A to frame B is

$$ {}_{b}\mathbf{f} = {}^{a}_{b}\mathbf{Ad}^{T}\ {}_{a}\mathbf{f}. \tag{2.26} $$

2.4.4 Newton-Euler Equations

The Newton-Euler equations for a rigid body can now be written in terms of the body's acceleration twist $\begin{pmatrix} \omega \\ v \end{pmatrix}$ and the wrench $\begin{pmatrix} \tau \\ f \end{pmatrix}$ acting on the body. To write these equations in a world-aligned frame at the center of mass, we simply rewrite the Newton and Euler equations (given in Section 2.3 into a matrix equation:

$$ \begin{pmatrix} \tau \\ f \end{pmatrix} = \begin{pmatrix} \mathcal{I} & 0 \\ 0 & m\mathbf{I} \end{pmatrix} \begin{pmatrix} \dot{\omega} \\ \dot{v} \end{pmatrix} + \begin{pmatrix} [\omega] & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathcal{I} & 0 \\ 0 & m\mathbf{I} \end{pmatrix} \begin{pmatrix} \omega \\ v \end{pmatrix}. \tag{2.27} $$

We shall use the symbol $\mathcal{M}$ for the mass-inertia matrix $\begin{pmatrix} \mathcal{I} & 0 \\ 0 & m\mathbf{I} \end{pmatrix}$. Matrix $\mathbf{R}$ here is the body's rotation matrix, and $\mathcal{I}$ is the angular inertia matrix in world coordinates (equal to $\mathbf{R}\mathcal{I}_{body}\mathbf{R}^T$).

In a frame "B" that is at the centre of mass and rotating with the body, the equation changes slightly.

$$ {}_b\mathbf{f} = \begin{pmatrix} \mathcal{I}_{body} & 0 \\ 0 & m\mathbf{I} \end{pmatrix} {}^b\dot{\mathbf{v}} + \begin{pmatrix} [{}^b\boldsymbol{\omega}] & 0 \\ 0 & [{}^b\boldsymbol{\omega}] \end{pmatrix} \begin{pmatrix} \mathcal{I}_{body} & 0 \\ 0 & m\mathbf{I} \end{pmatrix} {}^b\mathbf{v} \tag{2.28} $$

(See [MLS94] for a derivation of this.)

Throughout the rest of the thesis, we will write the Newton Euler equations as simply

$$ \mathbf{f} = \mathcal{M}\dot{\mathbf{v}} \tag{2.29} $$

The coriolis term (the second term on the right hand side in Equations 2.27 and 2.28) will be implicitly included in the wrench $\mathbf{f}$.

2.5 Constrained Dynamics

Most interesting simulations of rigid bodies involve some kind of constraints. Usually we want to model systems of bodies that are interacting in some way. Some bodies may be in contact with each other, or attached together by some type of joint. For example, we may wish to simulate a walking robot. We would model the robot as a system of rigid bodies connected by joints between the legs and the body, and there would be contact constraints between the legs and the ground. In this section we explain how to incorporate such constraints into a dynamic simulation.

2.5.1 Reduced Coordinates vs. Full Coordinates

There are two main approaches to incorporating constraints into a simulation. The first is the *reduced coordinate* approach, where the number of parameters describing the system's configuration is equal to the number of degrees of freedom in the system. In contrast to this, the *full coordinate* approach uses a full 6 degree of freedom parameterization of each body's position and velocity. We then represent the constraints by constraint equations, which restrict the position and velocity to a subspace of the possible values. Constraint forces are added to the system to keep the bodies on a trajectory that satisfies the constraint equations.

Reduced coordinates have certain advantages over full coordinates. In terms of computational efficiency, the reduced coordinate approach can be slightly faster for highly constrained systems where the remaining number of degrees of freedom is small. (Computation time for algorithms that use a full-coordinate approach tend to depend on the number of DOF that the constraints remove from the system, whereas algorithms for reduced coordinates depend on the number of DOF remaining in the system. However, this could be addressed by arranging the linear algebra carefully [LNPE92], [APC97]). Also, reduced coordinates have the advantage that the entire coordinate space corresponds to valid configurations. When using full coordinates, only some lower-dimensional subspace of the configuration space is consistent with the constraints. Numerical integration errors will lead to invalid states, making stabilization necessary (see Chapter 3). When using reduced coordinates, numerical error can make the simulation less accurate, but at least the resulting configuration will always be a plausible one[2].

On the other hand, reduced coordinates also have their drawbacks. In certain

[2]There is some danger in this too, of wrong results looking right

situations it may be unclear how to parameterize the configuration space. Kry [KP02] shows how a 5 degree of freedom parameterization can be used to model single point contact between curved surfaces. However, he points out that extending the reduced coordinate approach to more general, multiple contact situations is not so easy.

In general, the number of degrees of freedom changes when contacts are made and broken. Using full coordinates, we can handle these changes easily by introducing or removing constraint equations when necessary. This flexibility is one of our main motivations for choosing the full coordinate approach.

2.5.2 Expressing Constraints as Equations

We express constraints mathematically as algebraic matrix equations with position, velocity, or acceleration vectors as the unknowns. In general, the configuration space of a set of n rigid bodies has dimension $6n$. Adding constraint equations restricts the position (or velocity, or acceleration) to a subspace of smaller dimension.

The constraints discussed in this thesis will all be *holonomic* constraints, which means the constraint on the velocity can be found by taking the derivative of a position constraint. Constraints that do not fit this description are called *nonholonomic*. An example of a nonholonomic constraint is a ball rolling on a table without slipping. The position of the ball has five degrees of freedom, so there is only one degree of constraint (only the height is constrained). Taking the derivative of the position constraint would only give a velocity constraint of degree one. Additional constraints on the velocity are needed to prevent the ball from slipping.

We will describe position constraints with a *constraint function* $\boldsymbol{g}(\mathbf{p})$, which is a function from the space of possible positions of the rigid bodies, to $\mathbb{R}^d$, where

d is the number of degrees of freedom that the constraint removes from the system. If the constraint function returns a zero vector, then the position $\mathbf{p}$ satisfies the constraint. Constraint functions for specific kinds of constraints will be described in more detail in Chapter 4.

Position constraints can be divided into *equality constraints* (e.g., joint constraints), where the constraint is $\boldsymbol{g}(\mathbf{p}) = 0$, and *inequality constraints* (e.g., contact constraints), where $\boldsymbol{g}(\mathbf{p}) \geq 0$. For the rest of this section, we will just be talking about equality constraints. We will talk about contact constraints separately in the next section.

Constraining the position of an object also constrains its velocity. The velocity constraint function can be found by taking the time derivative of the constraint function. We want the velocity constraint function to be a linear function of the twists representing the velocities of all of the rigid bodies in the system. We can concatenate all these twists into a vector of twists $\mathbf{v} = \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_n \end{pmatrix}$. Velocity constraints are of the form

$$\frac{d\boldsymbol{g}_i}{dt} = \mathsf{J}_i\mathbf{v} = \begin{pmatrix} \mathbf{J}_{i1} & \mathbf{J}_{i2} & \dots & \mathbf{J}_{in} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} \boldsymbol{\omega}_1 \\ \boldsymbol{v}_1 \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{\omega}_2 \\ \boldsymbol{v}_2 \end{pmatrix} \\ \vdots \\ \begin{pmatrix} \boldsymbol{\omega}_n \\ \boldsymbol{v}_n \end{pmatrix} \end{pmatrix} = \mathbf{0}, \tag{2.30}$$

where the index i here is unique for each constraint. The matrix J is called the constraint's *Jacobian matrix*, which we refer to simply as the Jacobian[3]. The

[3]In Physics the Jacobian is the determinant of a matrix.

Jacobian is a function of the current position of the bodies involved in the constraint. We will go into detail on how to compute the Jacobians for various constraints in Chapter 4. In practice, most constraints will only involve one or two bodies, so most of the matrices $\mathbf{J}_1 \dots \mathbf{J}_n$ will be zero matrices.

A constraint on the acceleration can be found by taking the derivative again. Doing so results in an equation of similar form to Equation 2.30: $\mathsf{J}\dot{\mathbf{v}} + \boldsymbol{k} = \mathbf{0}$. The acceleration and velocity constraints use the same Jacobian matrix, but the acceleration constraint has an additional term $\boldsymbol{k}$ that depends on the velocities as well.

When there are many constraints, we will often write all constraints simultaneously. For a system with n bodies, and m constraints, our velocity constraint equation looks like

$$\frac{d\boldsymbol{g}}{dt} = \begin{pmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \vdots \\ \boldsymbol{g}_m \end{pmatrix} = \mathcal{J}\mathbf{v} = \begin{pmatrix} \mathsf{J}_1 \\ \vdots \\ \mathsf{J}_n \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_n \end{pmatrix} = \begin{pmatrix} \mathbf{J}_{11} & \mathbf{J}_{12} & \dots & \mathbf{J}_{1n} \\ \mathbf{J}_{21} & \mathbf{J}_{22} & & \mathbf{J}_{2n} \\ \vdots & & \ddots & \vdots \\ \mathbf{J}_{m1} & \mathbf{J}_{m2} & \dots & \mathbf{J}_{mn} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} \boldsymbol{\omega}_1 \\ \boldsymbol{v}_1 \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{\omega}_2 \\ \boldsymbol{v}_2 \end{pmatrix} \\ \vdots \\ \begin{pmatrix} \boldsymbol{\omega}_n \\ \boldsymbol{v}_n \end{pmatrix} \end{pmatrix} = \mathbf{0}. \tag{2.31}$$

Note that we refer to all of the matrices $\mathcal{J}$, J_i, and $\mathbf{J}_{ij}$ as Jacobians, but we use different fonts to distinguish between the different levels in this hierarchy:

- $\mathbf{J}$ is used for Jacobians that relate the velocity of a single body to a single constraint.
- J is used for Jacobians that relate the velocity of many bodies to a single constraint.

- $\mathcal{J}$ is used for Jacobians that relate the velocity of many bodies to many constraints.

2.5.3 Solving Constrained Dynamics Equations using Lagrange Multipliers

A constraint Jacobian matrix $\mathcal{J}$ has a dual use. In addition to relating the velocities to the rate of change of the constraint function $\boldsymbol{g}$, the rows of $\mathcal{J}$ act as basis vectors for constraint forces. Thus, when we solve for the constraint forces, we actually just need to solve for the coefficient vector $\boldsymbol{\lambda}$ (whose components are called the *Lagrange multipliers*) that contains the magnitudes of the forces that correspond to each of these basis vectors.

Solving Dynamics at the Force-Acceleration Level

The total force acting on the system is the sum of the external forces $\mathbf{f}_{ext}$ (in which we include Coriolis forces) and the constraint forces $\mathcal{J}^T\boldsymbol{\lambda}$. Combining the Newton-Euler equations,

$$\mathbf{f} = \mathcal{J}^T\boldsymbol{\lambda} + \mathbf{f}_{ext} = \mathcal{M}\dot{\mathbf{v}},$$

with the constraint equations $\mathcal{J}\dot{\mathbf{v}} + \boldsymbol{k} = 0$, we get the following system

$$\begin{pmatrix} \mathcal{M} & -\mathcal{J}^T \\ \mathcal{J} & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathbf{v}} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{ext} \\ -\boldsymbol{k} \end{pmatrix}, \tag{2.32}$$

which we can solve for the acceleration $\dot{\mathbf{v}}$ and the Lagrange multipliers $\boldsymbol{\lambda}$. Many authors have shown how to solve this system in linear time by exploiting the sparse structure of the matrix on the left hand side of Equation 2.32 [Ver74, LNPE92, APC97, Bar96].

There is no guarantee that the matrix in Equation 2.32 will be nonsingular. Singularity here is usually a result of over-constraining the bodies. In our hand made examples, we usually chose constraints carefully enough to avoid over-constraining. In general though, it would be nice to have a system that can handle singularities gracefully.

2.6 Incorporating Contact Constraints

Contact constraints are different from joint constraints, and require special treatment. There are a few important features that set contact constraints apart from other constraints:

- Contact forces can push bodies apart, but cannot pull bodies towards each other. This leads to inequalities in the constraint equations, whereas other types of constraints are equalities.
- If there are many points of contact between a pair of bodies, there may be many solutions for the contact forces that are plausible.
- In the presence of Coulomb friction, a solution to the dynamics equations may not exist.
- Contacts are transient – they come and go. Contacts can appear suddenly when bodies are moving towards each other, resulting in an impact.

Impacts between rigid bodies in reality result in large forces applied in over a very short time period, leading to sudden changes in velocity. Since the time scale of the impact is much smaller than the time scale of the overall simulation, it is

not practical to simulate the forces and accelerations involved. Instead, we simulate impacts with impulses (more on this in sections 2.7.3 and 4.4.2).

There are several methods for handling contact in rigid body simulations. Mirtich [Mir98] gives a summary of several different methods, and explains the advantages and disadvantages of each. We prefer the *Linear Complementarity Problem* approach, first introduced in the context of constrained mechanical systems by Lötstedt [L82], and later made popular in the computer graphics community by Baraff [Bar94].

To describe the contact model we use, we introduce the following terminology. We use the index i on symbols below to denote that the variable belongs to contact i.

- A *contact* consists of a pair of *contact points*, one point attached to one rigid body, and the other point attached to another rigid body. The contact points are sufficiently close together for our collision detection algorithm to report a collision.

- A *contact normal* is a unit vector that is normal to one or both of the surfaces at the contact points.

- The *contact constraint Jacobian* for constraint i, denoted by J_{c_i}, is a $1 \times 6n$ matrix, where n is the number of bodies (the subscript "c" here stands for "contact").

- A *contact force* $\mathsf{J}_{c_i}^T \lambda_{c_i}$ is a force acting in the direction of the contact normal which prevents the two rigid bodies from interpenetrating.

- The *separation distance* of a contact is the normal component of the distance

between the two contact points. It is negative when the bodies are interpenetrating at the contact. The constraint function $g_i(\mathbf{p})$ for a contact constraint returns the separation distance. The constraint is satisfied for $g_i \geq 0$.

- The *relative normal acceleration* a_i of a contact is the second derivative of the separation distance with respect to time ($a_i = \ddot{g}_i$). The acceleration constraint is satisfied when $a_i = \mathsf{J}_{c_i}\dot{\mathbf{v}} + k_i \geq 0$.

For the purposes of this section, assume that we have some separate method of dealing with impact. This section presents a method for finding the contact forces at more permanent contacts, which we will call *resting contacts.* We will define a resting contact as one where g and $\dot{g}$ are both zero (or nearly zero). For contacts where g or $\dot{g}$ are positive, no acceleration constraint is needed (these conditions indicate that the bodies are not touching, or moving apart at that contact, respectively). Situations where $\dot{g}$ is negative indicate impacts, which are resolved separately. The assumptions that at each resting contact $g = 0$ and $\dot{g} = 0$ are critical to the constraints that we enumerate below.

Let $\boldsymbol{a}$ be a vector containing the relative normal accelerations $\{a_1, ..., a_n\}$ for all contacts, and $\boldsymbol{\lambda}_c$ be a vector of contact force multipliers $\{\lambda_{c_1}, ..., \lambda_{c_n}\}$. The vectors $\boldsymbol{a}$ and $\boldsymbol{\lambda}_c$ are linearly related at a given $\mathbf{p}$. Additionally, we have three constraints:

1. The relative normal accelerations must be nonnegative: $\boldsymbol{a} = \mathcal{J}_c\dot{\mathbf{v}} + \boldsymbol{k} \geq \mathbf{0}$. Since g and $\dot{g}$ are zero, a negative acceleration would cause interpenetration.

2. The contact force magnitudes must be nonnegative (so as to push the bodies apart): $\boldsymbol{\lambda}_c \geq \mathbf{0}$

3. For each contact i, at least one of a_i, λ_{c_i} must be zero. We write this as $\boldsymbol{a}^T \lambda_c = 0$. This may not be obvious at first, but follows if we consider that there can only be a contact force if the bodies are actually touching ($g_i \lambda_{c_i} = 0$). Differentiating this twice using the chain rule, we get $\ddot{g}_i \lambda_{c_i} + 2\dot{g}_i \dot{\lambda}_{c_i} + g_i \ddot{\lambda}_i = 0$. Since we assume $g_i = 0$ and $\dot{g}_i = 0$, the second and third term cancel out leaving simply $\ddot{g}_i \lambda_{c_i} = 0$, and therefore one of $\ddot{g}_i, \lambda_{c_i}$ must be zero.

The problem of finding a solution to a linear equation given such constraints is called a *Linear Complementarity Problem* (LCP), a general class of problems that we discuss in Section 2.7.

If we let $\mathcal{J}_e$ be the Jacobian of equality constraints, and $\mathcal{J}_c$ be the Jacobian of contact constraints, we can extend Equation 2.32 to include the contact constraints as follows:

$$\begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{a} \end{pmatrix} - \begin{pmatrix} \mathcal{M} & -\mathcal{J}_e^T & -\mathcal{J}_c^T \\ \mathcal{J}_e & 0 & 0 \\ \mathcal{J}_c & 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathbf{v}} \\ \boldsymbol{\lambda}_e \\ \boldsymbol{\lambda}_c \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{ext} \\ -\boldsymbol{k}_e \\ -\boldsymbol{k}_c \end{pmatrix}, \tag{2.33}$$
$$\boldsymbol{a} \geq \mathbf{0}, \boldsymbol{\lambda}_c \geq \mathbf{0}, \boldsymbol{a}^T \boldsymbol{\lambda}_c = \mathbf{0}.$$

In this form, we have a *mixed LCP*, meaning that only some of the rows have complementarity constraints on them. To obtain a pure LCP, we must eliminate $\dot{\mathbf{v}}$ and $\boldsymbol{\lambda}_e$ by solving for those in terms of $\boldsymbol{\lambda}_c$. In Section 2.7.2 we explain how to convert a mixed LCP into a pure LCP.

2.7 Linear Complementarity Problems

A *Linear Complementarity Problem* (LCP) is, given a $n \times n$ matrix $\mathbf{M}$ and a n-vector $\boldsymbol{q}$, the problem of finding values for the variables $\{z_1, z_2, .., z_n\}$ and $\{w_1, w_2, .., w_n\}$ such that

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \mathbf{M} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} + \boldsymbol{q}, \tag{2.34}$$

and, for all i from 1 to n, $z_i \geq 0$, $w_i \geq 0$ and $z_i w_i = 0$. (These three constraints are called the *complementarity constraints*).

2.7.1 Lemke's Algorithm

Lemke's algorithm (also known as the *Complementary Pivot Algorithm*) is one of the most commonly used methods for solving LCPs. We describe the algorithm briefly here. Lemke's method is an iterative method that works by introducing an "artificial variable" z_0 so that Equation 2.34 becomes

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \mathbf{M} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} + \begin{pmatrix} z_0 \\ z_0 \\ \vdots \\ z_0 \end{pmatrix} + \boldsymbol{q}. \tag{2.35}$$

Introducing this artificial variable makes it easy to find an initial solution (we will explain why in a moment). After finding the initial solution, we iterate until we find a solution where $z_0 = 0$, and the artificial variable is no longer needed.

Rewriting Equation 2.35 so that all of the unknowns are grouped into a single vector, we have

$$\begin{pmatrix} -\boldsymbol{e}_n & -\mathbf{M} & \mathbf{I} \end{pmatrix} \begin{pmatrix} z_0 \\ \boldsymbol{z} \\ \boldsymbol{w} \end{pmatrix} = \boldsymbol{q}. \tag{2.36}$$

Here we use $\boldsymbol{e}_n$ to represent a vector containing n ones.

At any given iteration of the algorithm, only n of the variables from the set $\{z_0, z_1, ..., z_n, w_1, ..., w_n\}$ are *active* (i.e., nonzero). The other variables are *inactive* and are assumed to have a value of zero. The active variables are contained in a vector called the *basic vector*, $\boldsymbol{y}$. The *basis*, denoted by $\mathbf{B}$, is a matrix containing only the columns of the matrix $\begin{pmatrix} -\boldsymbol{e}_n & -\mathbf{M} & \mathbf{I} \end{pmatrix}$ that correspond to variables in the basic vector.

The invariant of the algorithm is that the basic vector always satisfies these properties:

1. There are always n variables in the basic vector.

2. There can be at most one variable from each *complementary pair* in the basic vector. A complementary pair is a pair $\{w_i, z_i\}$. This restriction steers us away from solutions where the complementarity constraint $z_i w_i = 0$ is not satisfied for all i.

3. The basis $\mathbf{B}$ corresponding to the basic vector is *feasible*, meaning that $\boldsymbol{y} = \mathbf{B}^{-1}\boldsymbol{q} \geq 0$ (i.e., if we set all the inactive variables to zero and we solve Equation 2.36 for the active variables, we have a solution where all active variables have a nonnegative value). This restriction steers us away from solutions where the constraints $w_i \geq 0$ and $z_i \geq 0$ are not satisfied for all i.

The idea of the algorithm is to swap variables in and out of the basic vector in way that ensures that the above properties always hold. For every iteration, except possibly the last one, the basic vector contains z_0, plus one variable from each of $n-1$ of the n complementary pairs. Hopefully, z_0 is eventually dropped out and another variable comes in, so that the basic vector contains one variable from each of the complementary pairs. When this happens, $\boldsymbol{y} = \mathbf{B}^{-1}\boldsymbol{q}$ is a solution to the LCP.

Initializing the Basic Vector

If all elements of $\boldsymbol{q}$ are non-negative, then the LCP is trivially solved with $\boldsymbol{w} = \boldsymbol{q}, \boldsymbol{z} = \mathbf{0}$. Otherwise, the algorithm is initialized with z_0 equal to the negative of the minimum element of $\boldsymbol{q}$. This way all elements of $z_0\boldsymbol{e_n} + \boldsymbol{q}$ are nonnegative.

Let t be the index of the minimum element in $\boldsymbol{q}$. To find the initial basic vector, we start with $\{w_1, ..., w_n\}$ and replace w_t with z_0.

For example, if $n = 4$, and q_3 is the minimum element of q, then our initial basic vector is $\{w_1, w_2, z_0, w_4\}$. Since all of the inactive variables are assumed to be zero, Equation 2.36 reduces to

$$\mathbf{B}\boldsymbol{y} = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ z_0 \\ w_4 \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix}. \tag{2.37}$$

By solving this system, we see that initializing the basic vector this way always leads to a feasible basis. Solving the third row first, we get $z_0 = -q_3$. We assumed that some the elements of $\boldsymbol{q}$ were nonnegative, and since q_3 is the minimum element, it must be negative. Thus z_0 is positive. Solving for the other variables,

we have $w_1 = q_1 - q_3$, $w_2 = q_2 - q_3$, and $w_4 = q_4 - q_3$. Since q_3 is less than q_1, q_2, and q_4, we are guaranteed a positive solution for w_1, w_2, and w_4.

Iterating the Algorithm

Each step of Lemke's algorithm begins by choosing an *entering variable* that will be inserted into the basic vector. In the initial iteration, we always choose z_t as this entering variable, where t is the index of the smallest element of $\boldsymbol{q}$. We choose z_t because $\{w_t, z_t\}$ is the only complementary pair that is left out of the basic vector. (We assume that as an initialization step, we have checked for the trivial case where all elements of q are positive. Thus at this stage we know that choosing w_t as the entering variable will not lead to a solution.)

The basic vector must always contain n variables, so in every iteration we must also have a *dropping variable.* We will choose the dropping variable so that the algorithm's invariant (number 3 in the list above) is preserved: if the dropping variable becomes inactive and the entering variable becomes active, there must still be a feasible (nonnegative) solution. We will temporarily imagine that the basic vector contains $n + 1$ variables so we can see how to choose the dropping variable.

Let $\mathbf{B}$ and $\boldsymbol{y}$ be the basis and basic vector from the previous iteration. Let x be the entering variable and $\boldsymbol{c}$ be the column of $\begin{pmatrix} -\boldsymbol{e}_n & -\mathbf{M} & \mathbf{I} \end{pmatrix}$ corresponding to the variable x. By introducing x, Equation 2.36 becomes

$$\begin{pmatrix} \mathbf{B} & \boldsymbol{c} \end{pmatrix} \begin{pmatrix} \boldsymbol{y} \\ x \end{pmatrix} = \boldsymbol{q}. \tag{2.38}$$

If x is given a nonzero value, how must the values of the variables in $\boldsymbol{y}$ change so that we still have a solution to our matrix equation?

$$
\begin{aligned}
\mathbf{B}(\mathbf{B}^{-1}\boldsymbol{q}) &= \boldsymbol{q}, \\
\mathbf{B}(\mathbf{B}^{-1}\boldsymbol{q}) - \boldsymbol{c}x + \boldsymbol{c}x &= \boldsymbol{q}, \\
\mathbf{B}((\mathbf{B}^{-1}\boldsymbol{q}) - (\mathbf{B}^{-1}\boldsymbol{c})x) + \boldsymbol{c}x &= \boldsymbol{q}, \\
\begin{pmatrix} \mathbf{B} & \boldsymbol{c} \end{pmatrix} \begin{pmatrix} (\mathbf{B}^{-1}\boldsymbol{q}) - (\mathbf{B}^{-1}\boldsymbol{c})x \\ x \end{pmatrix} &= \boldsymbol{q}.
\end{aligned} \tag{2.39}
$$

Imagine the value of x is increasing from zero. As we do this, some elements of $(\mathbf{B}^{-1}\boldsymbol{q}) - (\mathbf{B}^{-1}\boldsymbol{c})x$ will be increasing and some will be decreasing. The dropping variable is the first variable to reach zero. So, what is the smallest value of x such that some element of $(\mathbf{B}^{-1}\boldsymbol{q}) - (\mathbf{B}^{-1}\boldsymbol{c})x$ is zero? This is equivalent to finding the smallest positive element of $\mathbf{B}^{-1}\boldsymbol{q} \div \mathbf{B}^{-1}\boldsymbol{c}$, where "$\div$" is componentwise division. This is called the *minimum ratio test.*

Note that if some element of $\mathbf{B}^{-1}\boldsymbol{c}$ is nonpositive, then the corresponding element of $(\mathbf{B}^{-1}\boldsymbol{q}) - (\mathbf{B}^{-1}\boldsymbol{c})x$ will either increase or stay the same as we increase x. It will never reach zero, and is therefore not a candidate for the dropping variable. If all variables in $\boldsymbol{y}$ can be ruled out this way, then Lemke's algorithm fails to solve the LCP and we terminate the algorithm at this iteration.

If we can continue, we remove the dropping variable from the basic vector, and adjust the basis matrix accordingly. If z_0 is the dropping variable, then the we have found a solution to the LCP. Otherwise, we continue to the next iteration. The next iteration's entering variable is the complement of this iteration's dropping variable.

In degenerate situations, there may be a tie in the minimum ratio test. Arbitrarily breaking this tie may lead to infinite looping in Lemke's algorithm. To break the ties in a way that prevents looping, we use the *lexico-minimum ratio test*

[Mur88]. This can be summarized as follows:

- When there is a tie for minimum ratio in $\mathbf{B}^{-1}\boldsymbol{q} \div \mathbf{B}^{-1}\boldsymbol{c}$, we look for the smallest positive element in $\boldsymbol{v}_1 \div \mathbf{B}^{-1}\boldsymbol{c}$, where $\boldsymbol{v}_1$ is the first column in $\mathbf{B}^{-1}$.
- When there is a tie for minimum ratio in $\boldsymbol{v}_1 \div \mathbf{B}^{-1}\boldsymbol{c}$, look for the smallest positive element in $\boldsymbol{v}_2 \div \mathbf{B}^{-1}\boldsymbol{c}$, where $\boldsymbol{v}_2$ is the second column in $\mathbf{B}^{-1}$.
- And so on...

A more complete discussion of the Linear Complementarity Problem and Lemke's algorithm, refer to Cottle, Pang and Stone's book on LCP's [CPS92] (the definitive LCP reference), or Murty's book [Mur88], which provides examples and intuition and a good discussion of degeneracy and maintaining lexico-feasibility [4].

2.7.2 Converting a Mixed LCP to a pure LCP

Sometimes a linear complementarity problem must be solved simultaneously with other equations that do not have complementarity constraints. Such systems are called *Mixed LCPs*. In general, a mixed LCP has the form:

$$\begin{pmatrix} \mathbf{0} \\ \boldsymbol{w} \end{pmatrix} - \begin{pmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{pmatrix} = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix}, \tag{2.40}$$

$$\boldsymbol{w} \geq \mathbf{0}, \boldsymbol{z} \geq \mathbf{0}, \boldsymbol{w}^T\boldsymbol{z} = \mathbf{0}.$$

In this case, the variables in the vector $\boldsymbol{x}$ are unconstrained (they can be positive or negative), but the variables in $\boldsymbol{z}$ and $\boldsymbol{w}$ have complementarity constraints

[4] Murty's book is available online from
`http://ioe.engin.umich.edu/books/murty/linear_complementarity_webbook/`

on them. To obtain a pure LCP, as in Equation 2.34, we must eliminate $\boldsymbol{x}$. We assume that $\mathbf{P}$ is nonsingular. Rearranging the first row of Equation 2.40, we obtain

$$\boldsymbol{x} = \mathbf{P}^{-1}(-\boldsymbol{u} - \mathbf{Q}\boldsymbol{z}). \tag{2.41}$$

Then, substituting this into the second row, we have

$$\boldsymbol{w} + \mathbf{R}\mathbf{P}^{-1}\boldsymbol{u} + \mathbf{R}\mathbf{P}^{-1}\mathbf{Q}\boldsymbol{z} - \mathbf{S}\boldsymbol{z} = \boldsymbol{v}. \tag{2.42}$$

Reorganizing the terms gives us a pure LCP in the form of Equation 2.34.

$$\boldsymbol{w} - \underbrace{(\mathbf{S} - \mathbf{R}\mathbf{P}^{-1}\mathbf{Q})}_{\mathbf{M}}\boldsymbol{z} = \underbrace{\boldsymbol{v} - \mathbf{R}\mathbf{A}^{-1}\boldsymbol{u}}_{\boldsymbol{q}}. \tag{2.43}$$

After solving the LCP Equation 2.43, we can use Equation 2.41 to find $\boldsymbol{x}$.

Alternatively, rather than rearranging the mixed LCP into a pure LCP, it is fairly straightforward to modify Lemke's algorithm to deal directly with the mixed LCP. The modifications are:

1. All of the unconstrained variables are in the basic vector, and can never be swapped out for another variable.

2. The unconstrained variables are allowed to have negative values.

3. The unconstrained variables do not take part in the minimum ratio test.

2.7.3 Solvable LCPs for Contact with Friction

It is well known that that when Coulomb friction is added to the acceleration-level dynamics equations (Equation 2.33), the equations fail to have a solution in certain configurations, even for situations where there is only one contact involved.

Anitescu and Potra [AP97] present a time-stepping method that combines the acceleration-level LCP with an integration step for the velocities, arriving at a method having velocities and impulses as unknowns, rather than accelerations and forces. Their method is guaranteed to have a solution, regardless of the configuration and number of contacts.

To discretize the system 2.33, Anitescu and Potra approximate the acceleration as:

$$\dot{\mathbf{v}} \approx \frac{(\mathbf{v}_{t+h} - \mathbf{v}_t)}{h}, \tag{2.44}$$

where $\mathbf{v}_t$ and $\mathbf{v}_{t+h}$ are the velocities at the beginning of the current time step, and the next time step, respectively, and h is the time step size. We then arrive at the mixed LCP

$$\begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{a} \end{pmatrix} - \begin{pmatrix} \mathcal{M} & -\mathcal{J}_e^T & -\mathcal{J}_c^T \\ \mathcal{J}_e & 0 & 0 \\ \mathcal{J}_c & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{v}_{t+h} \\ \boldsymbol{\lambda}_e \\ \boldsymbol{\lambda}_c \end{pmatrix} = \begin{pmatrix} \mathcal{M}\mathbf{v}_t + h\mathbf{f}(\mathbf{p}_t, \mathbf{v}_t, t) \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}, \tag{2.45}$$

$$\boldsymbol{a} \geq \mathbf{0}, \boldsymbol{\lambda}_c \geq \mathbf{0}, \boldsymbol{a}^T \boldsymbol{\lambda}_c = \mathbf{0}.$$

Friction Constraints

Coulomb friction is introduced by adding some additional forces and constraints to the LCP 2.45.

In true Coulomb friction, the magnitude of the friction force must be less than μ times the magnitude of the normal force: $\|\boldsymbol{f}_f\| \leq \mu \|\boldsymbol{f}_n\|$. Geometrically speaking, the contact force is restricted to lie inside of a cone. However, to fit friction into our linear algebraic framework, it helps to approximate the friction

cone with a polyhedral cone. We do this by choosing a set of direction vectors $\boldsymbol{d}_{i1}, \boldsymbol{d}_{i2} \ldots \boldsymbol{d}_{in}$ tangent to contact i.

We express the frictional impulse for a contact i as:

$$\mathsf{J}_{f_i}^T \boldsymbol{\lambda}_{f_i} = \begin{pmatrix} \mathbf{d}_{i1} & \mathbf{d}_{i2} & \ldots & \mathbf{d}_{in} \end{pmatrix} \begin{pmatrix} \lambda_{f_{i1}} \\ \lambda_{f_{i2}} \\ \vdots \\ \lambda_{f_{in}} \end{pmatrix}, \tag{2.46}$$

where $\mathbf{d}_{i1}, \mathbf{d}_{i2} \ldots \mathbf{d}_{in}$ contain wrenches corresponding to equal and opposite forces on the contact points of the two contacting bodies, in directions $\boldsymbol{d}_{i1}, \boldsymbol{d}_{i2} \ldots \boldsymbol{d}_{in}$. The more directions we use for each contact, the better approximation to the friction cone we can get, at the expense of having to solve a larger LCP. Note that the set $\boldsymbol{d}_{i1}, \boldsymbol{d}_{i2} \ldots \boldsymbol{d}_{in}$ will contain vectors that are negatives of each other. This may seem redundant, but is necessary because of the nonnegativity constraints that we will place on the force multipliers when formulating the LCP.

The set of contact impulses lying within this polyhedral friction cone is then

$$\{\mathsf{J}_{c_i}^T \lambda_{c_i} + \mathsf{J}_{f_i}^T \boldsymbol{\lambda}_{f_i} \mid \lambda_{c_i} \geq 0,\ \boldsymbol{\lambda}_{f_i} \geq \mathbf{0},\ \boldsymbol{e}_i^T \boldsymbol{\lambda}_{f_i} \leq \mu_i \lambda_{c_i}\}, \tag{2.47}$$

where $\boldsymbol{e}_i = [1, 1, ..., 1]^T$, and μ_i is the coefficient of friction.

The complementarity constraints for each contact are as follows:

$$\begin{aligned} \gamma_i \boldsymbol{e}_i + \mathsf{J}_{f_i} \mathbf{v} \geq \mathbf{0} \quad & \text{complementary to} \quad \boldsymbol{\lambda}_{f_i} \geq \mathbf{0}, \\ \mu_i \lambda_{c_i} - \boldsymbol{e}_i^T \boldsymbol{\lambda}_{f_i} \geq 0 \quad & \text{complementary to} \quad \gamma_i \geq 0. \end{aligned} \tag{2.48}$$

If the normal impulse magnitude λ_{c_i} is zero, then the friction impulse must also be zero ($\boldsymbol{\lambda}_{f_i} = \mathbf{0}$), or the constraint $\mu_i \lambda_{c_i} - \boldsymbol{e}_i^T \boldsymbol{\lambda}_{f_i} \geq 0$ would be violated.

Whenever the normal impulse magnitude is positive, the variable γ_i ties the two friction constraints to each other. When there is no relative tangential motion at the contact, we have $\gamma_i = 0$, which allows the sum of the magnitudes of the friction impulses ($\boldsymbol{e}_i^T \boldsymbol{\lambda}_{f_i}$) to be less than $\mu_i \lambda_{c_i}$. However, if there is some relative tangential motion at the contact, then γ_i will be positive (it will be equal to the negative of the minimum element in $\mathsf{J}_{f_i}\mathbf{v}$). Having $\gamma_i > 0$ forces $\mu_i \lambda_{c_i} = \boldsymbol{e}_i^T \boldsymbol{\lambda}_{f_i}$, meaning that the friction impulse is at its maximum possible magnitude.

To summarize,

- if the contact impulse lies inside the friction cone (but not on its surface), then the bodies are not exhibiting relative tangential motion.
- If the bodies are in relative tangential motion, then the contact impulse lies on the surface of the friction cone, and exhibits negative work.
- As the approximation of the friction cone becomes closer to the ideal circular friction cone, the friction becomes closer to directly opposing the relative tangential motion.

These friction conditions are described more fully in [Ani97].[5]

Extending Equation 2.45 to include these friction constraints, we get the following linear complementarity problem

[5]However, we use different variable names here than Anitescu does

$$\begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{a} \\ \boldsymbol{\sigma} \\ \boldsymbol{\zeta} \end{pmatrix} - \begin{pmatrix} \mathcal{M} & -\mathcal{J}_e^T & -\mathcal{J}_c^T & -\mathcal{J}_f^T & 0 \\ \mathcal{J}_e & 0 & 0 & 0 & 0 \\ \mathcal{J}_c & 0 & 0 & 0 & 0 \\ \mathcal{J}_f & 0 & 0 & 0 & \mathbf{E} \\ 0 & 0 & \boldsymbol{\mu} & -\mathbf{E}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{v}_{t+h} \\ \boldsymbol{\lambda}_e \\ \boldsymbol{\lambda}_c \\ \boldsymbol{\lambda}_f \\ \boldsymbol{\gamma} \end{pmatrix} = \begin{pmatrix} \mathcal{M}\mathbf{v}_t + h\mathbf{f} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}, \tag{2.49}$$

$$\begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{\sigma} \\ \boldsymbol{\zeta} \end{pmatrix} \geq \mathbf{0}, \qquad \begin{pmatrix} \boldsymbol{\lambda}_c \\ \boldsymbol{\lambda}_f \\ \boldsymbol{\gamma} \end{pmatrix} \geq \mathbf{0}, \qquad \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{\sigma} \\ \boldsymbol{\zeta} \end{pmatrix}^T \begin{pmatrix} \boldsymbol{\lambda}_c \\ \boldsymbol{\lambda}_f \\ \boldsymbol{\gamma} \end{pmatrix} = \mathbf{0},$$

where $\boldsymbol{\mu} = \begin{pmatrix} \mu_1 & & \\ & \ddots & \\ & & \mu_n \end{pmatrix}$, and $\mathbf{E} = \begin{pmatrix} \boldsymbol{e}_1 & & \\ & \ddots & \\ & & \boldsymbol{e}_n \end{pmatrix}$.

2.7.4 Implicit Method for Stiff Systems

In a recent technical report, Anitescu and Potra [AP00] published an extension to the above, which we implemented in our system. Rather than using an explicit time-step in Equation 2.49, they suggest a linearly implicit method. This required only small modifications, yet the human animations we present in Chapter 5 would not have been possible without making this change (the feedback forces we used make the differential equations stiff, necessitating implicit integration) . The main requirement of this implicit method is that we must calculate the gradients of the stiff forces with respect to changes in the position and velocity of the rigid bodies.

To derive the linearly implicit method, we start with a backward Euler dis-

cretization of rigid body dynamics equation:

$$\mathcal{M}(\mathbf{v}_{t+h})\Big(\frac{\mathbf{v}_{t+h}-\mathbf{v}_t}{h}\Big) = \mathbf{f}\big(\mathbf{p}_{t+h}, \mathbf{v}_{t+h}, t+h\big). \tag{2.50}$$

The difficulty in solving this is that the force vector $\mathbf{f}$ is not known unless the position and velocity vectors $\mathbf{p}_{t+h}$ and $\mathbf{v}_{t+h}$ are known. In general, to apply backward Euler method to a nonlinear system, one must apply several Newton-Raphson iterations until a solution is found. We make a linear approximation using only the first Newton-Raphson iteration (hence the term "linearly implicit") that

$$\begin{aligned}&\mathbf{f}\big(\mathbf{p}_{t+h}, \mathbf{v}_{t+h}, t+h\big) \approx \\ &\quad \mathbf{f}\big(\mathbf{p}_t, \mathbf{v}_t, t\big) + \nabla_p \mathbf{f}\ h\ \mathbf{v}_{t+h} + \nabla_v \mathbf{f}\ \ (\mathbf{v}_{t+h} - \mathbf{v}_t),\end{aligned} \tag{2.51}$$

where $\nabla_p \mathbf{f}$ and $\nabla_v \mathbf{f}$ are the gradients of the function $\mathbf{f}$ with respect to change in position and velocity, respectively, and evaluated at $(\mathbf{p}_t, \mathbf{v}_t, t)$. If we substitute this into Equation 2.50 and move all of the terms with $\mathbf{v}_{t+h}$ to the left hand side, we obtain

$$\begin{aligned}&(\mathcal{M} - h^2\nabla_p \mathbf{f} - h\nabla_v \mathbf{f})\mathbf{v}_{t+h} = \\ &\quad \mathcal{M}\mathbf{v}_t - h\nabla_v \mathbf{f}\mathbf{v}_t + h\ \mathbf{f}\big(\mathbf{p}_t, \mathbf{v}_t, t\big).\end{aligned} \tag{2.52}$$

It is convenient to use the notation

$$\widehat{\mathcal{M}} = \mathcal{M} - h^2\nabla_p \mathbf{f} - h\nabla_v \mathbf{f}, \tag{2.53}$$

and

$$\widehat{\mathbf{f}} = \mathbf{f}\big(\mathbf{p}_t, \mathbf{v}_t, t\big) - \nabla_v \mathbf{f}\mathbf{v}_t, \tag{2.54}$$

so that the linearly implicit equation for forward dynamics closely resembles the time-stepping dynamics equations we have seen already, such as Equation 2.49. The implicit version is:

$$\begin{pmatrix} \widehat{\mathcal{M}} & \mathcal{J}^T \\ \mathcal{J} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{v}_{t+h} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} -\mathcal{M}\mathbf{v}_t - h\widehat{\mathbf{f}} \\ 0 \end{pmatrix}. \tag{2.55}$$

In our implementation, we estimate the force gradients $\nabla_p \mathbf{f}$ and $\nabla_v \mathbf{f}$ numerically, by evaluating $\mathbf{f}$ for several position and velocity values in the neighborhood of the current state of the system.

Chapter 3

Stabilization For Simulation With Contact

3.1 Overview of Stabilization

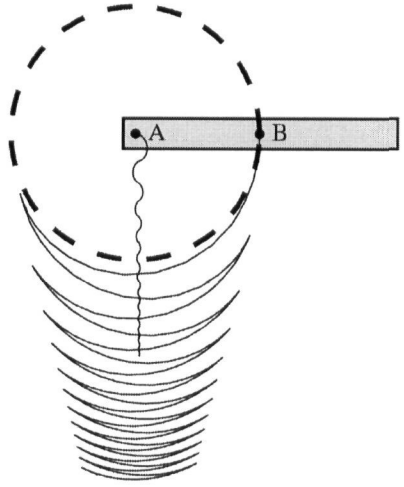

Figure 3.1: An unstabilized simulation of a swinging pendulum.

Figure 3.1 is an example situation that motivates the need for stabilization.

The figure shows a simple two-dimensional simulation of a swinging pendulum: one rigid body with one constraint. The only external force is that of gravity. The constraint is a hinge joint constraint that anchors point A to a fixed point. Over time, however, point A actually drifts, due to numerical integration errors. The centre of mass, rather than staying on the circular path that it is supposedly constrained to (shown with a dashed line), continues to fall further from that path as time progresses.

We can lessen this problem by using higher order, more accurate integration scheme (the ones used here are only first-order accurate[1]), or by using smaller time steps. Even with first-order methods, the local error is proportional to the square of the step size, so decreasing the step size to one tenth effectively decreases the local error to one hundredth. However, even using the best integration methods and small time steps, numerical drift can never be completely eliminated. Furthermore, it may not always be feasible to use the more complicated integrators or decrease the step size and still achieve acceptable performance. Even if we cannot achieve very high numerical accuracy, we still wish to have simulations that are plausible – ones where the constraints are always met. This section describes methods for counteracting constraint drift using stabilization.

Before continuing, we need to define our problem more clearly. *Stability* is a word that is used with a variety of meanings in different contexts in the differential equation literature. By the word *stabilization* here, we mean *stabilization of an ODE with respect to an invariant set.* We shall now explain what this means.

The position constraint equation coupled with the constrained Newton-Euler equations (see Equation 2.32) together are an example of a *differential-algebraic*

[1]see [AP98] for a precise definition of the accuracy of an integration scheme.

equation (DAE). This is a term for a system of equations containing both differential equations and algebraic equations. DAEs are a relatively new area of research, and the methods for solving them directly are somewhat difficult. The usual approach is to convert the DAE into an equivalent ordinary differential equation (ODE) that can be solved by conventional techniques. In our case, we can do this by differentiating the position constraint equations twice, to arrive at a constraint in terms of accelerations, then substituting this acceleration constraint into the Newton-Euler equations. In doing this, we arrive at an ODE. The approach we use is actually slightly different, as we have described in Section 2.7.3.[2]

The penalty of this approach is that we lose the constraints on the position and velocity. Upon discretization of the ODE, we introduce numerical errors, and we will experience *drift* away from the *constraint manifold.* The constraint manifold (also known as the *invariant set*) is the subset of the state space in which the constraints are satisfied. To counteract this, we must stabilize the ODE with respect to the invariant set. In other words, we must alter the ODE so that it has the same solutions as the original whenever $\boldsymbol{g}(\mathbf{p}) = 0$, but whenever $\boldsymbol{g}(\mathbf{p}) \neq 0$ the solutions is attracted towards the invariant set.

In this chapter we will introduce two methods of stabilization, in the context of rigid body simulation. First we will discuss *Baumgarte* stabilization, a method that is very popular because of its simplicity. We will then introduce *Post-Stabilization*, which is based on the work of Ascher et al. [Asc97] [ACR94]

[2]We use *time-stepping* methods, where a numerical integration step is built into the system of equations we solve, and the equations are given in terms of velocities and impulses, rather than forces and accelerations.

3.2 Baumgarte Stabilization

Baumgarte's stabilization technique [Bau72] is one of the most familiar and commonly used methods, because of its simplicity. The idea here is to replace the acceleration constraint equation $\ddot{\boldsymbol{g}} = \mathcal{J}\dot{\mathbf{v}} + \boldsymbol{k} = 0$ with some a linear combination of the acceleration, velocity and position constraint equations:

$$0 = \ddot{\boldsymbol{g}} + \alpha\dot{\boldsymbol{g}} + \beta\boldsymbol{g}, \tag{3.1}$$

which creates a more stable ODE. If the velocity and position constraints are satisfied, the last two terms on the right hand side vanish, and we are left with the original acceleration constraint equation. A physical interpretation of this method is that we are adding in additional correction forces, proportional to the error in the velocity and position constraints, to counteract drift.

Because our implementation uses only velocity-level constraints rather than acceleration constraints (see Section 2.7.3), we use an even simpler version of Baumgarte stabilization where we replace the velocity constraint $\dot{\boldsymbol{g}} = \mathcal{J}\mathbf{v} = 0$ with $\dot{\boldsymbol{g}} + \alpha\boldsymbol{g} = \mathcal{J}\mathbf{v} + \alpha\boldsymbol{g} = 0$.

The main difficulty of using Baumgarte stabilization is that it is not always easy to find an appropriate value for the constants α and β.

3.3 Post-Stabilization

Another approach to stabilization is to follow each integration step with a stabilization step. The stabilization step takes the result of the integration step as input, and gives a correction so that the end result is closer to the constraint manifold. Post-stabilization methods are discussed in detail and compared to other stabi-

lization methods by Ascher et al. [ACR94]. Here we give one interpretation of post-stabilization that fits into Ascher's broader definition.

Let $\mathbf{p}$ be the position of a set of rigid bodies after the integration step. Let g be the constraint function (see Section 2.5.2). In general, due to numerical drift, $g(\mathbf{p}) \neq \mathbf{0}$. Let $\mathbf{G} = \frac{\partial g}{\partial \mathbf{p}}$. In our stabilization step, we wish to find some $\mathbf{dp}$ such that $g(\mathbf{p} + \mathbf{dp}) = 0$. Assuming $\mathbf{dp}$ will be small, we can make the approximation that

$$g(\mathbf{p} + \mathbf{dp}) \approx g(\mathbf{p}) + \mathbf{G}(\mathbf{p})\mathbf{dp}. \tag{3.2}$$

Rearranging this, we see that the stabilization term $\mathbf{dp}$ should satisfy

$$\mathbf{G}\mathbf{dp} = -g(\mathbf{p}). \tag{3.3}$$

In general, $\mathbf{G}$ is not square, so $\mathbf{G}^{-1}$ does not exist. One way to solve for $\mathbf{dp}$ is to use the pseudoinverse of $\mathbf{G}$:

$$\mathbf{dp} = -(\mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1})g(\mathbf{p}). \tag{3.4}$$

This idea works fine as long as $\mathbf{G}\mathbf{G}^T$ is nonsingular, which is usually the case for a system that contains only equality constraints. However, we often run into singularities when contact constraints are involved. This is because the collision detector may find many contact points between a pair of objects, leading to constraints that are redundant. One approach to deal with singularities is to use a pseudoinverse formula based on singular value decomposition of $\mathbf{G}$. By truncating the small (nearly zero) singular values, we can find a pseudoinverse of $\mathbf{G}$ even when $\mathbf{G}\mathbf{G}^T$ is singular.

Using a singular value decomposition in each time step, however, would be prohibitively expensive, and it does not take the inequality constraints involved

with contact into account. Instead, we find it natural to pose the post-stabilization problem as an LCP, just as we do with the dynamics equations. We explain this method in the next section.

3.4 Fitting Post-Stabilization into Our Dynamics LCP Framework

As mentioned above, in the absence of contact constraints, we can find the stabilization term using the pseudoinverse of $\mathbf{G}(\mathbf{p})$ (Equation 3.4). Doing so is equivalent to solving the system

$$-\begin{pmatrix} \mathbf{I} & -\mathbf{G}(\mathbf{p})^T \\ \mathbf{G}(\mathbf{p}) & 0 \end{pmatrix} \begin{pmatrix} \mathbf{dp} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ g(\mathbf{p}) \end{pmatrix}. \tag{3.5}$$

In other words, we can express the problem of finding the post-stabilization step in terms as a problem of finding Lagrange multipliers, just as we did with the dynamics equations (see Equation 2.32). We can make 3.5 look more like Equation 2.32 by doing two things:

1. Replace $\mathbf{G}(\mathbf{p})$ with $\mathcal{J}(\mathbf{p})$. These two matrices are both constraint Jacobians. The difference is that $\mathbf{G}$ multiplies with changes in position (7-vectors consisting of a translation, plus a quaternion), whereas $\mathcal{J}$ multiplies with twists, which are 6-vectors. It makes more sense to use $\mathcal{J}$ because we would already have code to compute it from our implementation of the dynamics computations. As a result of using $\mathcal{J}$, the post-stabilization step $\mathbf{dp}$ would be a twist and needs to be converted back into our position representation, using Equation 2.9.[3]

[3]Note that in two-dimensional rigid body simulations, $\mathbf{G}$ and $\mathcal{J}$ are identical, so this

2. Replace the identity matrix with the mass matrix $\mathcal{M}$. By doing so, we are no longer using the pseudoinverse $\mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1}$, but a weighted pseudoinverse, $\mathcal{M}^{-1}\mathbf{G}^T(\mathbf{G}\mathcal{M}^{-1}\mathbf{G}^T)^{-1}$. This corresponds to favouring the position change that requires the least amount of energy. This way, for example, a rotation around an axis with low angular inertia would be favoured over a rotation around an axis with high angular inertia.

Making these two changes gives us

$$-\begin{pmatrix} \mathcal{M} & -\mathcal{J}(\mathbf{p})^T \\ \mathcal{J}(\mathbf{p}) & 0 \end{pmatrix} \begin{pmatrix} \mathbf{dp} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ g(\mathbf{p}) \end{pmatrix}. \tag{3.6}$$

Similar to our dynamics equations for systems with contact, it is natural to place complementarity constraints on the variables having to do with the contact constraints.

- The post-step should never pull contacting bodies towards each other at the contact points, only push them apart: $\boldsymbol{\lambda}_c \geq \mathbf{0}$.
- If contact constraint functions were evaluated after adding the post step, the result must not be negative, but may be positive: $\boldsymbol{g}_c^+ = (\boldsymbol{g}_c^- + \mathcal{J}_c\mathbf{dp}) \geq 0$ (we use superscripts "-" and "+" to denote "before" and "after" the post-step. That is, $\boldsymbol{g}_c^- = \boldsymbol{g}(\mathbf{p})$, and $\boldsymbol{g}_c^+$ approximates $\boldsymbol{g}(\mathbf{p} + \boldsymbol{dp})$).
- $\boldsymbol{\lambda}_c^T \boldsymbol{g}_c^+ = \mathbf{0}$: This constraint roughly means that for each contact i, either we are pushing the bodies apart at i or contact i's constraint will be satisfied in the absence of any push at i.

somewhat confusing distinction can be ignored.

Adding the stabilization for contact constraints, we have the LCP

$$\begin{pmatrix} 0 \\ 0 \\ \boldsymbol{g}_c^+ \end{pmatrix} - \begin{pmatrix} \mathcal{M} & -\mathcal{J}_e^T & -\mathcal{J}_c^T \\ \mathcal{J}_e & 0 & 0 \\ \mathcal{J}_c & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{dp} \\ \boldsymbol{\lambda}_e \\ \boldsymbol{\lambda}_c \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \boldsymbol{g}_e^- \\ \boldsymbol{g}_c^- \end{pmatrix}, \tag{3.7}$$

$$\boldsymbol{g}_c^+ \geq \mathbf{0}, \boldsymbol{\lambda}_c \geq \mathbf{0}, \boldsymbol{\lambda}_c^T \boldsymbol{g}_c^+ = \mathbf{0}.$$

Chapter 4

System Architecture

We have implemented a rigid body simulation system using Java. We chose Java for its extensive freely available libraries (including the linear algebra package *Colt* [Hos], which we used heavily), its support for multi-threading, its cross-platform compatibility, support for object-oriented programming, and ease of use. Our implementation goal was to build a general purpose, extensible, interactive simulation system that could handle simulations of constrained rigid bodies with contact. Writing this software was no easy task. Similar systems have been developed by others, both for research and for commercial products in the entertainment industry. However, source codes are generally not available to the public, and it is difficult to find material that goes into the specifics of implementing a general purpose rigid body simulator. A *SIGGRAPH* tutorial by Baraff [Bar97] provides some details, as does Cremer's thesis [Cre89]. In this chapter we will try to explain the important implementation details of our simulator, so that others will not have to feel as though they are reinventing the wheel, as the author sometimes did.

4.1 System Overview

Figure 4.1 shows a flow chart of the main loop of the simulator. We describe the steps here in more detail.

1. The time step begins with the rigid bodies in position $\mathbf{p}(t)$ and moving with velocity $\mathbf{v}(t)$. We have a set of equality constraints, and a set of contact constraints.

2. We total up the external forces by calling the `apply` method of each `ExternalForce` object.

3. We calculate the Jacobian matrices and the mass matrix based on the current position $\mathbf{p}(t)$.

4. We solve the forward dynamics LCP to determine the velocity for the next time step, $\mathbf{v}(t+h)$.

5. We use a numerical integrator to find the next position $\mathbf{p}(t+h)$. (Higher-order numerical methods may actually solve the LCP in step 4 several times).

6. We pass the position vector $\mathbf{p}(t+h)$ to the collision detector. This position vector may represent a state where there are interpenetrations between bodies. We assume that at the beginning of the time step there were no interpenetrations, and therefore if we interpolate the positions between the beginning and end of the time step, we will find a position free of interpenetration. We repeatedly cut the step size in half until we find such a state. The revised step size is dt.

7. The collision detector returns a set of contacts. We create new contact constraints from these contacts.

8. We update the Jacobians based on this new position vector $\mathbf{p}(t + dt)$.

9. We calculate the constraint functions for all of the constraints. This is input to the post-stabilization. After taking our post-step, we have some revised position vector $\mathbf{p}_r$, for which the constraints are met. We then progress on to the next time step.

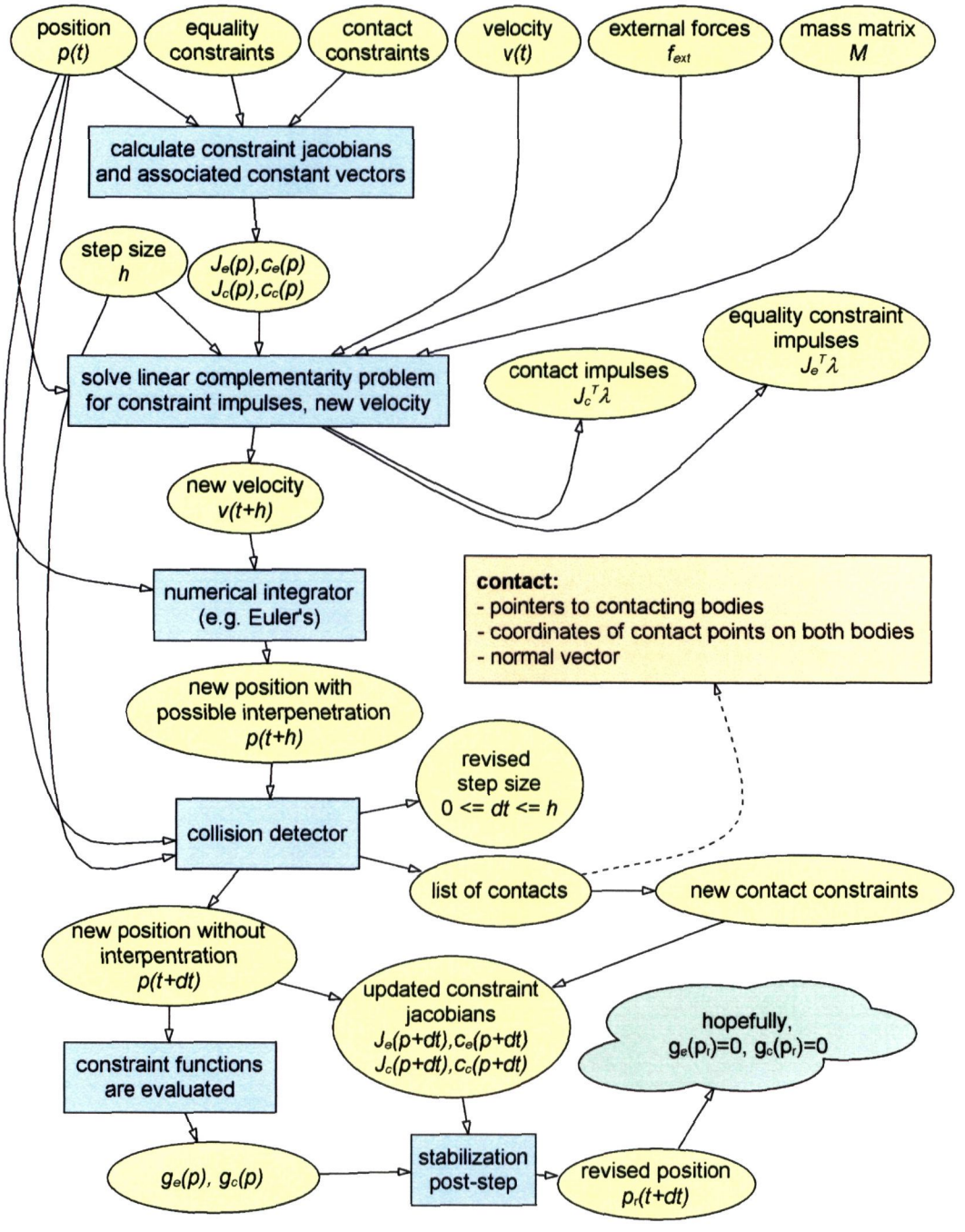

Figure 4.1: Overview of the simulator.

4.2 Accommodating both 2D and 3D Simulations

Although three dimensional simulations are clearly better than two dimensional ones in terms of realism, there are good reasons for wanting to simulate 2D rigid body motion. In certain situations, the motion in one dimension may be unimportant. For instance, in a head-on car crash simulation, we may choose to ignore the left-right motion of the crash dummy, and just simulate the crash in 2D using a side-view of the dummy. In a simulation of an air-hockey game, we could probably neglect the height dimension and just use a 2D bird's-eye view of the table.

Two dimensional simulations are very useful from a software development point of view. We often found it much easier to understand the bugs and problems of our system by testing it on 2D examples. There are a couple of reasons for this. The first reason is that it is 2D simulations are much more easily visualized than 3D simulations. Debugging 3D simulations involves moving the viewpoint around a lot so that the area of interest becomes clearer. For example, in 3D it is difficult to visually tell whether two objects are separated, in contact, or in interpenetration. If two objects are close to each other, the only way to get an idea of how far apart they are is to manoeuver the camera so that the crack between them is visible. In 2D, there is no such problem. The other benefit of 2D simulations for debugging is that the matrices and vectors involved are much smaller, and therefore easier to verify by hand, if necessary. One of the challenges of writing a piece of software like this is information overload on the programmer during debugging sessions, and anything that can be done to alleviate this is usually welcome.

Fortunately, it is easy to allow both 2D and 3D rigid body simulations within the same code, as long as care is taken not to hard-code any assumptions about the length of position and velocity vectors, or the width of mass and Jacobian matrices.

4.3 Keeping Track of Simulation State

Most of the state information needed during the simulation is contained within a single `RigidBodySet` object, which keeps track of the following values:

- The positions and velocities of all rigid bodies in the set
- The mass-inertia matrix $\mathcal{M}$ for the set of rigid bodies
- The Jacobian $\mathcal{J}_e$ for the equality constraints affecting the system
- The Jacobian $\mathcal{J}_c$ for the contact constraints, and $\mathcal{J}_f$ for the friction constraints.
- The matrices $\mathbf{E}$ and $\boldsymbol{\mu}$, also used in friction computations
- Pointers to all `Constraint` objects
- Pointers to all `RigidBody` objects

The `RigidBodySet` class encapsulates all of the matrix and vector values needed to solve the forward dynamics problem (see Equation 2.49). Although we could have distributed this information among many objects (for instance, keeping each body's position in a separate `RigidBody` object), this centralized approach is more efficient because it reduces the amount of copying needed when formulating the dynamics problem. Conveniently, our matrix package allows us to create *sub-range views* of matrices and vectors, so that the information pertaining to individual `RigidBody` and `Constraint` objects can be mirrored inside those objects without creating a separate copy of the data. Figure 4.2 is a visual representation of some of our data structures. In this instance, there are three rigid bodies connected into a chain. The arrows in this diagram represent pointers, and the boxes are vectors and matrices.

4.4 Implementing Constraints

In Section 2.5.2, we described how constraints are represented mathematically. Recall that a constraint on the positions of rigid bodies is represented by a *constraint function* $\boldsymbol{g}(\mathbf{p})$, and velocity constraints are given as linear equations of the form $\mathsf{J}(\mathbf{p})\ \mathbf{v} = \mathbf{0}$. Thus, every constraint object must implement methods to calculate $\boldsymbol{g}(\mathbf{p})$ and $\mathsf{J}(\mathbf{p})$.

4.4.1 Joint Constraints

Constraint Equations for a Ball Joint

A ball joint (sometimes called a "ball-and-socket" joint) constrains a point $\boldsymbol{p}_a$ on body A and a point $\boldsymbol{p}_b$ on body B to lie at the same location. The constraint function $\boldsymbol{g}(\mathbf{p})$ should then return zero if and only if $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ are at the same place. The obvious choice then, is to have $\boldsymbol{g}$ return the vector from one point to the other:

$$\boldsymbol{g}(\mathbf{p}) = \boldsymbol{p}_a - \boldsymbol{p}_b. \tag{4.1}$$

Recall (from Equation 2.21) that the velocity of a point $\boldsymbol{p}$ attached to a rigid body is given by

$$\dot{\boldsymbol{p}} = [V]\boldsymbol{p} = \begin{pmatrix} [\boldsymbol{\omega}] & \boldsymbol{v} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}, \tag{4.2}$$

where $V = \begin{pmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{pmatrix}$ is the body's velocity given in twist coordinates. The easiest frame to calculate the Jacobian in is a frame having its origin at the location of the

joint. Let us call this frame the "joint frame". If ${}^{j}\boldsymbol{V}_a$ is the velocity of body A given as twist coordinates in the joint frame, and ${}^{j}\boldsymbol{V}_b$ is the velocity of body B, then the constraint is simply:

$$ {}_{j}\mathbf{J}_a {}^{j}\boldsymbol{V}_a + {}_{j}\mathbf{J}_b {}^{j}\boldsymbol{V}_b = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^{j}\boldsymbol{\omega}_a \\ {}^{j}\boldsymbol{v}_a \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} {}^{j}\boldsymbol{\omega}_b \\ {}^{j}\boldsymbol{v}_b \end{pmatrix} = \mathbf{0}. \tag{4.3} $$

Since our implementation uses a velocity representation where the linear velocity component is the velocity of the centre of mass, we must transform the Jacobians so that they are expressed in frames with their origins at the bodies' centres of mass. We do this by expressing the above equation in terms of these frames. Let frame A be a frame at the centre of mass of body A, and frame B be a frame at the centre of mass of body B. Both of these frames are rotated to align with the world frame, and for simplicity we will assume that the joint frame J is also aligned with the world frame (the rotation of frame J does not affect Equation 4.3). Equation 4.3 becomes:

$$ {}_{j}\mathbf{J}_a \, {}^{j}_{a}\mathbf{Ad} \, {}^{a}\boldsymbol{V}_a + {}_{j}\mathbf{J}_b \, {}^{j}_{b}\mathbf{Ad} \, {}^{b}\boldsymbol{V}_b = 0. \tag{4.4} $$

From this, we can see that the coordinates of Jacobian $\mathbf{J}_a$ in frame A are

$$ {}_{a}\mathbf{J}_a = {}_{j}\mathbf{J}_a \, {}^{j}_{a}\mathbf{Ad} = \begin{pmatrix} \mathbf{0} & \mathbf{I}_3 \end{pmatrix} \begin{pmatrix} \mathbf{I}_3 & \mathbf{0} \\ [\boldsymbol{r}_a] & \mathbf{I}_3 \end{pmatrix} = \begin{pmatrix} -[\boldsymbol{r}_a] & \mathbf{I}_3 \end{pmatrix} = \begin{pmatrix} 0 & r_{a_z} & -r_{a_y} & 1 & 0 & 0 \\ -r_{a_z} & 0 & r_{a_x} & 0 & 1 & 0 \\ r_{a_y} & -r_{a_x} & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.5} $$

The vector $\boldsymbol{r}_a$ is the vector from the joint to the centre of mass of body A. Doing a similar transformation to ${}_b\mathbf{J}_b$ gives

$$ {}_b\mathbf{J}_b = {}_j\mathbf{J}_b \, {}^j_b\mathbf{Ad} = \begin{pmatrix} 0 & -r_{b_z} & r_{b_y} & -1 & 0 & 0 \\ r_{b_z} & 0 & -r_{b_x} & 0 & -1 & 0 \\ -r_{b_y} & r_{b_x} & 0 & 0 & 0 & -1 \end{pmatrix}. \tag{4.6} $$

Hinge Joints

A hinge joint is a one degree of freedom joint, meaning that it constrains five degrees of freedom. It can be seen as an extension of the ball joint, just with two extra constraints added to ensure that the bodies can only rotate relative to each other around the axis of rotation. For example, if the axis of rotation of the joint is the X axis of frame J, and ${}^j\boldsymbol{\omega}_a$ and ${}^j\boldsymbol{\omega}_b$ are the angular velocities of bodies A and B given in world coordinates, then the constraints are

$$ {}^j\boldsymbol{\omega}_{a_y} - {}^j\boldsymbol{\omega}_{b_y} = 0, \qquad {}^j\boldsymbol{\omega}_{a_z} - {}^j\boldsymbol{\omega}_{b_z} = 0. \tag{4.7} $$

Appending these constraints on to the ball joint constraint equation (4.3), gives us the hinge joint constraint equation:

$$ {}_j\mathbf{J}_a \, {}^j\boldsymbol{V}_a + {}_j\mathbf{J}_b \, {}^j\boldsymbol{V}_b = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^j\boldsymbol{\omega}_a \\ {}^j\boldsymbol{v}_a \end{pmatrix} + \begin{pmatrix} 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} {}^j\boldsymbol{\omega}_a \\ {}^j\boldsymbol{v}_a \end{pmatrix} = \mathbf{0}. \tag{4.8} $$

In terms of our twists ${}^a\boldsymbol{V}_a$ and ${}^b\boldsymbol{V}_b$ that we use in our implementation, the above constraint is equivalent to the constraint

$$\begin{pmatrix} {}^a j_x & {}^a j_y & {}^a j_z & 0 & 0 & 0 \\ {}^a k_x & {}^a k_y & {}^a k_z & 0 & 0 & 0 \\ 0 & r_{a_z} & -r_{a_y} & 1 & 0 & 0 \\ -r_{a_z} & 0 & r_{a_x} & 0 & 1 & 0 \\ r_{a_y} & -r_{a_x} & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^a\boldsymbol{\omega}_a \\ {}^a\boldsymbol{v}_a \end{pmatrix} + \begin{pmatrix} -{}^b j_x & -{}^b j_y & -{}^b j_z & 0 & 0 & 0 \\ -{}^b k_x & -{}^b k_y & -{}^b k_z & 0 & 0 & 0 \\ 0 & -r_{b_z} & r_{b_y} & -1 & 0 & 0 \\ r_{b_z} & 0 & -r_{b_x} & 0 & -1 & 0 \\ -r_{b_y} & r_{b_x} & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} {}^b\boldsymbol{\omega}_b \\ {}^b\boldsymbol{v}_b \end{pmatrix} = \mathbf{0}, \tag{4.9}$$

where $\boldsymbol{j}$ and $\boldsymbol{k}$ are two mutually orthogonal vectors that are also orthogonal to the hinge joint's axis of rotation. We will call these vectors the *forbidden axes*, because the bodies are forbidden to rotate with respect to each other around these axes.

Universal Joint

A joint with two degrees of rotational freedom is sometimes called a universal joint. Universal joints have two axes of rotation, one which is fixed to a first body, and another which is fixed to a second body. The two axes of rotation are usually orthogonal to each other. The Jacobian of a universal joint is very similar to the hinge joint's Jacobian, except it has only four rows because there is only one forbidden axis about which the relative rotation of the two bodies is disallowed.

Constraint Functions for Hinge Joints and Universal Joints

Since hinge joints and universal joints are extensions of ball joints, the first three rows of the Jacobian are the same between these three types of joints. Correspondingly,

three of the elements in the vector returned by the constraint function $\boldsymbol{g}(\mathbf{p})$ should be the same as the ball joint's constraint function, given in Equation 4.1.

The remaining elements of the constraint function are measures of rotational error. In the universal joint, there is just one forbidden axis, so there is one remaining element of $\boldsymbol{g}(\mathbf{p})$ that returns the relative rotation (in radians) around that forbidden axis. We calculate this by decomposing the relative rotation $\mathbf{R}_a^{-1}\mathbf{R}_b$ (where $\mathbf{R}_a$ is the rotation matrix for body A, and $\mathbf{R}_b$ is the rotation matrix for body B) into three rotations:

$$\mathbf{R}_a^{-1}\mathbf{R}_b = rot(\boldsymbol{v}_3, \gamma) \times rot(\boldsymbol{v}_2, \beta) \times rot(\boldsymbol{v}_1, \alpha), \tag{4.10}$$

where $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are the rotation axes, and $\boldsymbol{v}_3$ is the forbidden axis (the coordinates of these axes are given in body A's coordinate frame here). Then γ is our rotational error measure that we return as the one remaining element of $\boldsymbol{g}(\mathbf{p})$.

We use a similar method for hinge joints, except that there are two forbidden axes and only one rotation axis. Then in Equation 4.10, $\boldsymbol{v}_1$ is the rotation axis, and we return β and γ as the two rotational elements of the constraint function $\boldsymbol{g}(\mathbf{p})$.

When defining the Jacobian of a constraint, we said that $\frac{d\boldsymbol{g}}{dt} = \mathsf{J}\mathbf{v}$. Note that when using the constraint function described here and the constraint Jacobians described above for the hinge joint and universal joint, it is not exactly true in general that $\frac{d\boldsymbol{g}}{dt} = \mathsf{J}\mathbf{v}$, but it is a good approximation when the constraint's rotational error is small. We could derive a more exact J, but the derivation is rather complicated to go into here, and we find that the Jacobians we gave work fine in practice.

One question that comes up when implementing this is how to deal with constraint error in the initial configuration of the bodies. For example, let's say that bodies A and B are constrained by a hinge joint so that they can only rotate relative

to each other about the Z-axis of body A. What if, in the initial configuration given by the user, the relative rotation $\mathbf{R}_a^{-1}\mathbf{R}_b$ is some rotation about the Y-axis? There are two ways we can deal with this:

1. We can decompose $\mathbf{R}_a^{-1}\mathbf{R}_b$ as in Equation 4.10. Doing this, the initial configuration will in violation of the constraint, and the post-stabilization will correct for it so that the relative rotation between the two bodies is purely around the Z-axis.

2. We can assume that the initial rotations of the bodies (which we will call $\mathbf{R}_{a_0}$ and $\mathbf{R}_{b_0}$) do not violate the rotational constraints, and interpret the rotation constraint to mean that the relative rotation $\mathbf{R}_a^{-1}\mathbf{R}_b$ must be of the form $rot_z(\theta) \times (\mathbf{R}_{a_0}^{-1}\mathbf{R}_{b_0})$, where $rot_z(\theta)$ is the matrix for a rotation of theta about the Z-axis. In this method, rather than decomposing $\mathbf{R}_a^{-1}\mathbf{R}_b$ into three angles, we decompose $(\mathbf{R}_a^{-1}\mathbf{R}_b)(\mathbf{R}_{a_0}^{-1}\mathbf{R}_{b_0})^{-1}$. This matrix can be seen as the change in relative rotation between the initial configuration and the current configuration.

We have implemented the constraint functions using the latter method, because it allows the user to have more flexibility in specifying the initial rotations of the objects.

4.4.2 Contact Constraints

In Section 2.6, we defined a contact as a pair of *contact points*, one point attached to one rigid body, and the other point attached to another rigid body, and the *separation distance* of a contact is the normal component of the distance between the two contact points. The constraint function $\boldsymbol{g}(\mathbf{p})$ for a contact constraint returns

the separation distance, minus ϵ:

$$g(\mathbf{p}) = \boldsymbol{n} \cdot (\boldsymbol{x}_a - \boldsymbol{x}_b) - \epsilon, \tag{4.11}$$

where $\boldsymbol{n}$ is the contact normal, $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ are the contact points on bodies A and B, respectively, and ϵ is the *contact tolerance.* Thus, if the separation distance is less than the contact tolerance, the constraint function returns a negative number, indicating that something is wrong.

To derive the Jacobian of the contact constraint, we will make use of three coordinate frames. W is the "world" coordinate frame. A and B are world-aligned frames with origins at the centres of mass of the two bodies. Because the axes of all three of these coordinate frames are aligned, a 3-vector can be transformed from frame to frame without changing the coordinates:

$$\begin{aligned} {}^{w}\dot{\boldsymbol{x}}_a &= {}^{a}\dot{\boldsymbol{x}}_a, \\ {}^{w}\dot{\boldsymbol{x}}_b &= {}^{b}\dot{\boldsymbol{x}}_b. \end{aligned} \tag{4.12}$$

The contact point locations are given by

$$\begin{aligned} \boldsymbol{x}_a &= \boldsymbol{p}_a + \boldsymbol{r}_a, \\ \boldsymbol{x}_b &= \boldsymbol{p}_b + \boldsymbol{r}_b. \end{aligned} \tag{4.13}$$

By assuming the contact points stay fixed with respect to the bodies, and that the contact normal stays fixed, we can approximate the rate of change of the separation distance g (and thus derive the Jacobian of the contact constraint) by applying the Equation 2.8 to both of the contact points[1]:

[1]The normal, in general, also changes . We currently do not take this into account. See [Can86] for a derivation of $\dot{\boldsymbol{n}}$

$$
\begin{aligned}
\frac{d}{dt}\boldsymbol{g}(\mathbf{p}) &\approx \boldsymbol{n}\cdot(\dot{\boldsymbol{x}}_a - \dot{\boldsymbol{x}}_b),\\
&= {}^{w}\boldsymbol{n}\cdot({}^{w}\dot{\boldsymbol{x}}_a - {}^{w}\dot{\boldsymbol{x}}_b),\\
&= {}^{w}\boldsymbol{n}\cdot({}^{a}\dot{\boldsymbol{x}}_a - {}^{b}\dot{\boldsymbol{x}}_b),\\
&= {}^{w}\boldsymbol{n}\cdot(({}^{a}\boldsymbol{v}_a + [{}^{a}\boldsymbol{\omega}_a]{}^{a}\boldsymbol{x}_a) - ({}^{b}\boldsymbol{v}_b + [{}^{b}\boldsymbol{\omega}_b]{}^{b}\boldsymbol{x}_b)),\\
&= {}^{w}\boldsymbol{n}\cdot(({}^{a}\boldsymbol{v}_a - [{}^{a}\boldsymbol{x}_a]{}^{a}\boldsymbol{\omega}_a) - ({}^{b}\boldsymbol{v}_b - [{}^{b}\boldsymbol{x}_b]{}^{b}\boldsymbol{\omega}_b)),\\
&= (({}^{w}\boldsymbol{n}^T \quad {}^{a}\boldsymbol{v}_a - \underbrace{{}^{w}\boldsymbol{n}^T[{}^{a}\boldsymbol{x}_a]{}^{a}\boldsymbol{\omega}_a}_{=({}^{a}\boldsymbol{x}_a]^T{}^{w}\boldsymbol{n})^T \quad {}^{a}\boldsymbol{\omega}_a = (-{}^{a}\boldsymbol{x}_a\times{}^{w}\boldsymbol{n})^T \quad {}^{a}\boldsymbol{\omega}_a}) - ({}^{w}\boldsymbol{n}^T \quad {}^{b}\boldsymbol{v}_b - {}^{w}\boldsymbol{n}^T[{}^{b}\boldsymbol{x}_b]{}^{b}\boldsymbol{\omega}_b)),\\
&= \underbrace{\begin{pmatrix}({}^{a}\boldsymbol{x}_a\times{}^{w}\boldsymbol{n})^T & {}^{w}\boldsymbol{n}^T\end{pmatrix}}_{{}_a\mathbf{J}_a}\begin{pmatrix}{}^{a}\boldsymbol{\omega}_a\\ {}^{a}\boldsymbol{v}_a\end{pmatrix} + \underbrace{\begin{pmatrix}-({}^{b}\boldsymbol{x}_b\times{}^{w}\boldsymbol{n})^T & -{}^{w}\boldsymbol{n}^T\end{pmatrix}}_{{}_b\mathbf{J}_b}\begin{pmatrix}{}^{b}\boldsymbol{\omega}_b\\ {}^{b}\boldsymbol{v}_b\end{pmatrix}.
\end{aligned}
\tag{4.14}
$$

The Jacobians of the individual contact constraints are each only one row, representing the fact that each contact constraint removes only one degree of freedom. In each time step of the simulation, after running collisions detection, we create a single matrix $\mathcal{J}_c$, where each row of $\mathcal{J}_c$ is the Jacobian matrix of one contact.

Resting Contact vs. Impact

Resting contacts and impact contacts differ in that with resting contact, the relative normal velocity at the contact is close to zero, while in the case of impacts, the bodies are moving towards each other, so the relative normal velocity will be negative. Most objects have a certain amount of bounciness, so we expect to see the objects move apart after an impact, whereas there is no bounce involved in resting contact. Simulating impact in an acceleration-level dynamics system (see Equation 2.33) is

tricky because simpler models of impacts usually involve impulsive forces – infinite accelerations over an infinitesimal amount of time. However, since we use a time-stepping method (Equation 2.49) where velocities and impulses are the variables, our implementation can naturally handle both resting contact and impact using the same model. We use the very simple Newton impact model:

$$v^+ = -k_{restitution}\, v^-, \tag{4.15}$$

where v^- is the relative velocity of the two bodies before impact, v^+ is the relative velocity after impact, and $k_{restitution}$ is the *coefficient of restitution.* Although this model is typically used to describe single contact collisions between particles, we apply it to rigid body dynamics by insisting that for each contact, the separation velocity of the contact after impact is greater than or equal to $-k_{restitution}$ times the separation velocity of the contact before impact, leading to the following velocity constraint equation:

$$\underbrace{\mathbf{J}_a \begin{pmatrix} \boldsymbol{\omega}_a \\ \boldsymbol{v}_a \end{pmatrix} + \mathbf{J}_b \begin{pmatrix} \boldsymbol{\omega}_b \\ \boldsymbol{v}_b \end{pmatrix}}_{\mathbf{JV}} + \underbrace{k_{restitution}\left(\mathbf{J}_a \begin{pmatrix} \boldsymbol{\omega}_a^- \\ \boldsymbol{v}_a^- \end{pmatrix} + \mathbf{J}_b \begin{pmatrix} \boldsymbol{\omega}_b^- \\ \boldsymbol{v}_b^- \end{pmatrix}\right)}_{\boldsymbol{c}} \geq \mathbf{0}. \tag{4.16}$$

Admittedly, this impact model is a bit of an abuse of physics. The Newton impact model is not intended to model collisions involving multiple contacts, and the impact law we use here has been proven to be physically incorrect. In particular, the Newton impact model has been shown to generate energy under certain types of frictional collisions [Str91]. More theoretically sound impact models have been proposed (see Stronge [Str91], and Ullrich [Ull98]), but even very complicated models are still inaccurate at predicting real impacts.

Despite the simplicity of our impact model, we were satisfied that it produced plausible-looking impacts and we did not feel there was much to gain by exploring other models. However, better impact models would certainly be necessary if the simulator was being used in an application where the computed impact responses were expected to be a good prediction of real-world behaviour (e.g., simulating crash tests of cars to improve their design).

Friction Jacobians

The friction Jacobian J_f of a contact is similar in nature to the contact constraint Jacobian J_c:

- $\mathsf{J}_c\mathbf{v}$ gives the relative *normal* velocities at each of the contacts. $\mathsf{J}_f\mathbf{v}$ gives the relative *tangential* velocities at each of the contacts.
- The column of J_c^T is a wrench that corresponds to a force being applied at the contact, in the *normal* direction. The columns of J_f^T give wrenches for forces applied tangentially at the contact.

Thus, J_c and J_f have a mathematically similar form. As we saw above, for an individual contact constraint, the two Jacobians are

$$ {}_a\mathbf{J}_{ca} = \begin{pmatrix} ({}^a\boldsymbol{x}_a \times {}^w\boldsymbol{n})^T & {}^w\boldsymbol{n}^T \end{pmatrix}, \qquad {}_b\mathbf{J}_{cb} = \begin{pmatrix} ({}^b\boldsymbol{x}_b \times {}^w\boldsymbol{n})^T & {}^w\boldsymbol{n}^T \end{pmatrix}. \tag{4.17} $$

The friction Jacobians are

$$ {}_a\mathbf{J}_{fa} = \begin{pmatrix} ({}^a\boldsymbol{x}_a \times {}^w\boldsymbol{d}_1)^T & {}^w\boldsymbol{d}_1^T \\ ({}^a\boldsymbol{x}_a \times {}^w\boldsymbol{d}_2)^T & {}^w\boldsymbol{d}_2^T \\ \vdots & \vdots \\ ({}^a\boldsymbol{x}_a \times {}^w\boldsymbol{d}_n)^T & {}^w\boldsymbol{d}_n^T \end{pmatrix}, \qquad {}_b\mathbf{J}_{fb} = \begin{pmatrix} ({}^b\boldsymbol{x}_b \times {}^w\boldsymbol{d}_1)^T & {}^w\boldsymbol{d}_1^T \\ ({}^b\boldsymbol{x}_b \times {}^w\boldsymbol{d}_2)^T & {}^w\boldsymbol{d}_2^T \\ \vdots & \vdots \\ ({}^b\boldsymbol{x}_b \times {}^w\boldsymbol{d}_n)^T & {}^w\boldsymbol{d}_n^T \end{pmatrix}, \quad (4.18) $$

where $\{\boldsymbol{d}_1, \boldsymbol{d}_2, ...\boldsymbol{d}_n\}$ is a set of directions that are tangential to the surface.

4.5 Collision Detection

Although collision detection is a very important part of any rigid body simulation, our research has not focused on the collision detection problem. Instead, we have essentially viewed the collision detector as a black box, so our software is not tied to any particular collision detection system. However, some collision detectors are more well-suited to our purposes than others. In particular, many collision detection packages only return the closest pair of points between the two objects, or just the distance between them. This is insufficient for modelling contact between two planes that are resting on each other. Ideally, we want the collision detector to return enough contacts so that plane-plane and edge-plane contacts can be modelled properly. The collision detector that we use (a sphere-tree based collision detector, like [Qui94]) returns contacts for all of the following situations:

- The distance between a vertex on object A and a face on object B is less than ϵ.
- The distance between a vertex on object B and a face on object A is less than ϵ.

- The distance between an edge on object A and an edge on object B is less than ϵ.

- The distance between a vertex on object A and an edge on object B is less than ϵ.

The variable ϵ is the *contact tolerance*. Each one of these situations results in a `Contact` object being created. The contact object contains pointers to the two rigid bodies that are in contact, coordinates of the contact points on body A and B in body-relative coordinates, a normal vector, and the separation distance at that contact. This information is used to generate a contact constraint.

One problem with returning contacts for all of the situations mentioned above is that we may get far more contacts than we need. For example, imagine a simulation of a small, highly tesselated spherical ball being hit by a bat. Since the ball and bat are both convex shapes, we really only need one contact point to simulate the contact event, but it is likely that many contacts will be detected. The problem is worsened if the contact tolerance ϵ is large. Having a large number of contacts will cause the LCP solver to run slowly and the simulation to bog down. This is a difficult problem to deal with, and one that remains unsolved in our system.

4.6 External Forces

Things like gravity, springs or muscles between bodies, motors, etc. are implemented as external forces. External forces in general are a function of the position and velocity of the bodies they are attached to. The `ExternalForce` object implements this function. In addition to this, some `ExternalForce` objects also implement functions that return the gradient of the force with respect to changes in position

and velocity. This information is needed for our implicit time-stepping method (see Section 2.7.4). This is important in external forces that change quickly for small changes in position or velocity. For example, when two rigid bodies are attached to each other by a stiff spring, the spring will resist being extended by a small amount with a large force.

4.7 Interacting with the Simulation

User interaction in the system was accomplished in two ways. The first is through a mouse-controlled *attractor-repulsor* external force object. The object was represented on the screen as a small spherical cursor that could repositioned by moving the mouse. Motion was normally in the same plane as the ground, but one could move the sphere vertically by dragging with the middle mouse button. Holding the left mouse button down caused the sphere to emit a repulsive force that pushed objects away. The right mouse button would cause an attractive force. The magnitudes of these forces would increase as the sphere got closer to a rigid body.

Another way we allowed the user to interact was by allowing them to throw a rigid ball into the scene. Again, the user would use the mouse to control this. When the user clicked, the ball would be thrown along the ray from the viewpoint, through the pixel on the near-plane that the user clicked on.

4.8 LCP Solver Implementation

4.8.1 Exploiting Coherence Between Basis Matrix of Consecutive Iterations

For the most part, implementing Lemke's algorithm is straightforward. One point that may not be so obvious is how to avoid recomputing the inverse of the basis matrix from scratch in each iteration of the algorithm. Recall from our description of Lemke's algorithm (Section 2.7.1) that in each iteration, we remove one variable from the basic vector and add another variable. This corresponds to changing one row in the basis. Fortunately, there are ways to take advantage of the fact that only one row is changing in this matrix, meaning we do not have to run an expensive $O(n^3)$ matrix inversion in every iteration.

To exploit this coherence between iterations, we use the Sherman-Morrison-Woodbury (SMW) formula [GvL89]. This formula gives an expression for the inverse of a matrix after undergoing some change, given the inverse of the original matrix. The general form of the SMW formula takes a matrix $\mathbf{A}_0$ and its inverse $\mathbf{A}_0^{-1}$ as input, and returns the inverse of matrix $\mathbf{A}_s = \mathbf{A}_0 + \mathbf{R}\mathbf{S}^T$. The formula is:

$$\mathbf{A}_s^{-1} = \mathbf{A}_0^{-1} - \mathbf{A}_0^{-1}\mathbf{R}\left[\mathbf{I} + \mathbf{S}^T\mathbf{A}_0^{-1}\mathbf{R}\right]^{-1}\mathbf{S}^T\mathbf{A}_0^{-1}. \tag{4.19}$$

In the case we are interested in, $\mathbf{A}_s$ and $\mathbf{A}_0$ differ only in one column, which we will call column i. We will say that $\boldsymbol{v}_{old}$ is the i'th column of $\mathbf{A}_0$, and $\boldsymbol{v}_{new}$ is the i'th column of $\mathbf{A}_s$. Then let $\boldsymbol{\Delta v} = \boldsymbol{v}_{new} - \boldsymbol{v}_{old}$. The expression relating $\mathbf{A}_s$ to $\mathbf{A}_0$ is then

$$\mathbf{A}_s = \mathbf{A}_0 + \boldsymbol{\Delta v} \underbrace{\begin{pmatrix} 0 & 0 & \dots & 1 & \dots & 0 & 0 \end{pmatrix}}_{\text{only the } i'th \text{ element is one, rest are zero}} . \tag{4.20}$$

If we use $\boldsymbol{\Delta v}$ as $\mathbf{R}$ in Equation 4.19, and $\begin{pmatrix} 0 & 0 & \dots & 1 & \dots & 0 & 0 \end{pmatrix}$ as $\mathbf{S}^T$, then $\mathbf{S}^T\mathbf{A}_0^{-1}$ is simply the i'th row of $\mathbf{A}_0^{-1}$ (we'll use the symbol $\boldsymbol{b}^T$ to denote this row vector). Then the SMW formula simplifies to

$$\mathbf{A}_s^{-1} = \mathbf{A}_0^{-1} - \Big(\frac{\mathbf{A}_0^{-1}\boldsymbol{\Delta v}}{1 + \boldsymbol{b}^T\boldsymbol{\Delta v}}\Big)\boldsymbol{b}^T. \tag{4.21}$$

Using this formula, the inverse of the basis can be updated in $O(n^2)$ time in each iteration of Lemke's algorithm, rather than the $O(n^3)$ time that a general matrix inversion would take.

4.8.2 A Generalization of Baraff's Technique for Incorporating the Auxiliary Constraints into a Linear Time Lagrange Multiplier Approach

In David Baraff's paper *Linear-Time Dynamics Using Lagrange Multipliers* [Bar96], he shows an approach for exploiting the sparseness when solving for the Lagrange multipliers in the constrained dynamics equations. One section of this paper deals with how to incorporate "auxiliary" constraints, such as contact constraints, which must be solved externally using an LCP solver. It is interesting to note that Baraff's technique can be generalized to any mixed LCP system, and the notation actually becomes much simpler in this generalization. The generalized version is as follows.

Let us assume that the matrix $\mathbf{P}$ in the mixed LCP 2.40 is sparse, and we know of an algorithm *sparsefactor* to factor it in linear time, as well as an algorithm *sparsesolve* to solve a system $\mathbf{P}\boldsymbol{x} = \boldsymbol{b}$ in linear time. In this case, it

would be wasteful to explicitly invert $\mathbf{P}$ when using Equation 2.43 to LCP to a pure LCP. Instead, we can save effort by using our sparse solution algorithms to find $\mathbf{P}^{-1}\boldsymbol{u}$ and $\mathbf{P}^{-1}\mathbf{Q}$ without computing $\mathbf{P}^{-1}$ itself. The steps to the algorithm are:

1. Run *sparsefactor* to factor $\mathbf{P}$.

2. Run *sparsesolve* to find $\boldsymbol{x}^{base}$ such that $-\mathbf{P}\boldsymbol{x}^{base} = \boldsymbol{u}$. Then $\boldsymbol{x}^{base} = \mathbf{P}^{-1}\boldsymbol{u}$. This $\boldsymbol{x}^{base}$ vector is the solution to the unconstrained variables $\boldsymbol{x}$ of the LCP, in absence of the constrained variables $\boldsymbol{z}$.

3. Next we compute the matrix $\mathbf{P}^{-1}\mathbf{Q}$, which tells us how changes in $\boldsymbol{z}$ affect $\boldsymbol{x}$. Let $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_k$ be the columns of the matrix $\mathbf{Q}$. We can use *sparsesolve* k times to individually calculate the vectors $\mathbf{P}^{-1}\boldsymbol{b}_1, \mathbf{P}^{-1}\boldsymbol{b}_2, \ldots, \mathbf{P}^{-1}\boldsymbol{b}_k$. Concatenating all these into one matrix gives $\mathbf{P}^{-1}\mathbf{Q}$.

4. $\mathbf{P}^{-1}\boldsymbol{u}$ and $\mathbf{P}^{-1}\mathbf{Q}$ can now be plugged into Equation 2.43. We solve this pure LCP using Lemke's algorithm to determine the value for $\boldsymbol{z}$.

5. We run *sparsesolve* once more to calculate $\boldsymbol{x}$, using the relationship $\mathbf{P}\boldsymbol{x} = (-\boldsymbol{u} - \mathbf{Q}\boldsymbol{z})$.

4.8.3 Numerical Difficulties We Encountered with Lemke's Algorithm

We found implementing Lemke's algorithm to be a bit troublesome due to unexpected numerical problems. The problems mainly had to do with the minimum ratio test, or the lexico-minimum ratio test. To recall, the minimum ratio test is where we search for the minimum positive element of $\mathbf{B}^{-1}\boldsymbol{q} \div \mathbf{B}^{-1}\boldsymbol{c}$, where $\mathbf{B}$ is the basis matrix, $\boldsymbol{c}$ is the basis column of the entering variable, and $\boldsymbol{q}$ is one of the

original inputs to the problem. As one of the algorithm invariants, the elements in the vector $\mathbf{B}^{-1}\boldsymbol{q}$ are always all non-negative. Thus, if we are looking for the minimum positive element of $\mathbf{B}^{-1}\boldsymbol{q} \div \mathbf{B}^{-1}\boldsymbol{c}$, we can exclude all ratios i for which $(\mathbf{B}^{-1}\boldsymbol{c})_i \leq 0$. It turns out that this is not quite sufficient. Due to numerical error, It can happen that there is some i for which both $(\mathbf{B}^{-1}\boldsymbol{q})_i$ and $(\mathbf{B}^{-1}\boldsymbol{c})_i$ are extremely small positive numbers, and the ratio between them is smaller than any of the other ratios. In this case (which happened to us surprisingly often), this i is mistakenly chosen by the minimum ratio test. We have addressed this problem by replacing the test $(\mathbf{B}^{-1}\boldsymbol{c})_i \leq 0$ with $(\mathbf{B}^{-1}\boldsymbol{c})_i \leq \epsilon$ for some tiny ϵ that is large with respect to precision of the floating-point calculations.

Another issue is that of ties in the minimum ratio test. As we discussed in Section 2.7.1, these ties can be resolved by using the lexico-minimum ratio test. However, sometimes in the presence of numerical error, there is a "near-tie", where the tying ratios differ by some very small amount. In this case, should we break the tie in favour of the minimum ratio, or pretend that the ratios are equal and move on to the next level of the lexico-minimum ratio test? The answer to this is not clear to us.

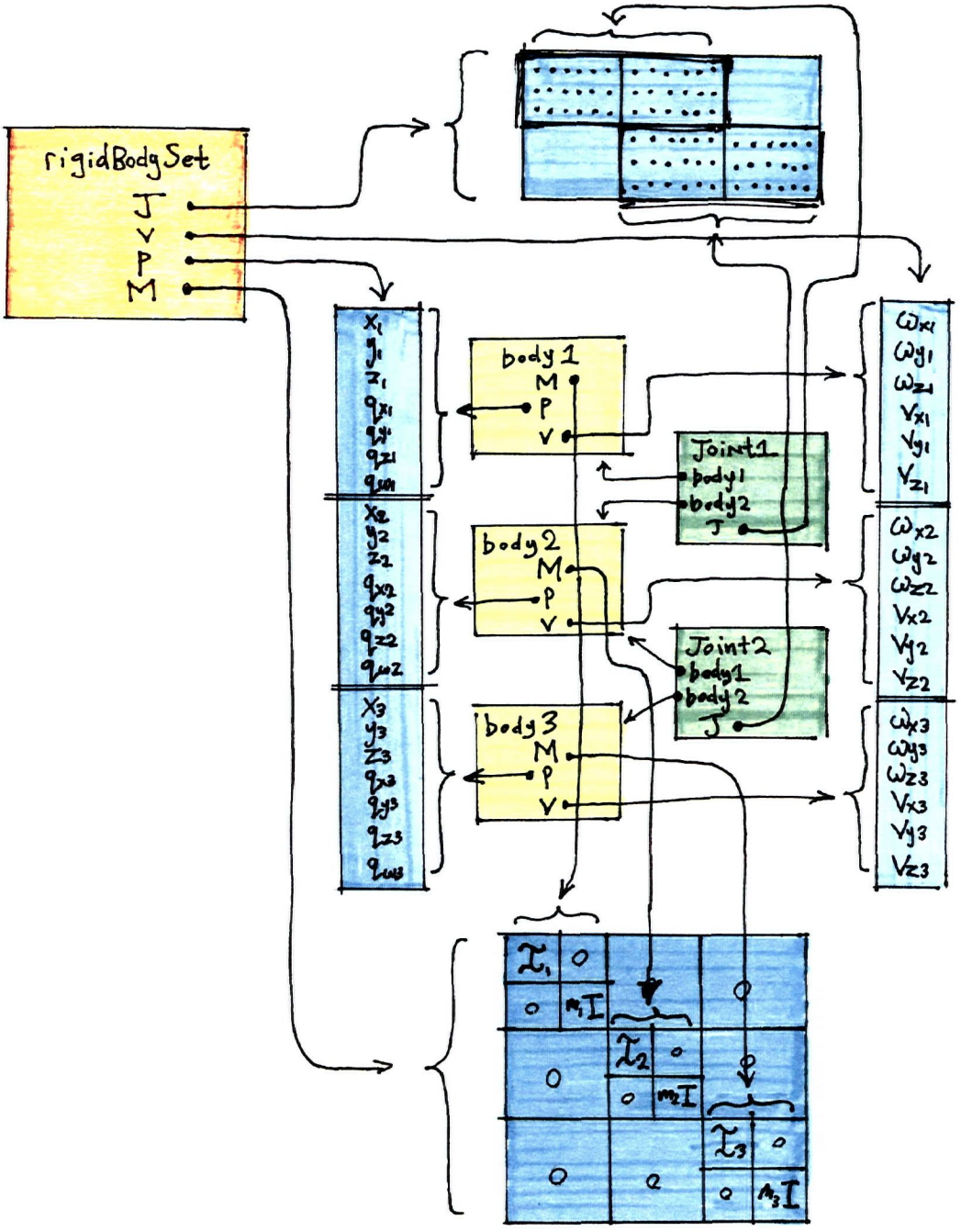

Figure 4.2: An example showing the relationship between an instance of `RigidBodySet` and several of `RigidBody` objects and `Constraint` objects. The `p`, `v`, `M` and `Je` variables of the `RigidBodySet` point to large vectors and matrices, while the `p`, `v`, `M` and `Je` variables of the `RigidBodys` and `Constraints` point to smaller *sub-range views* of these large matrices.

Chapter 5

Results and Discussion

This chapter presents the results of our work. Two of the goals of our simulation software were to be extensible, and to run at interactive rates. To demonstrate that these goals were met, we show an application of our software to interactive human character animation. We also present some results from experiments that test the post-stabilization method discussed in Chapter 3.

5.1 An Application: Transforming Motion-Capture Data

In this section, we describe an application of our rigid body simulator for animation of human characters in interactive virtual environments. We will explain some extensions we made to the simulator so that motion capture data could be used to drive physical simulations of humans which would react in physically plausible ways to external physical stimuli.

Motion capture is an increasingly popular method for creating complex animations, especially for human motions. This involves recording kinematic information, usually as a set of joint angles sampled at regular time intervals. Using

kinematic data to directly animate the character is problematic in an interactive simulation such as a sports video game, because it is difficult to make it look like the character is really interacting with its environment. Physical constraints do not affect the character when the motions are just playbacks of the kinematic data. In situations where these physical constraints are important, we are better off using a dynamic physical model of the character and driving the model with forces that are derived from the motion capture data.

We use a biologically-based motor control model to drive the character. We discuss the biomechanics background more extensively in a technical report [CYP02]. To summarize our approach, we model the character's muscle forces as a combination of a feedback term, which models the elasticity of the muscles, and a feedforward term, which is computed using inverse dynamics.

Using our technique, motion sequences can be modified at interactive rates so that they react to unexpected disturbances. We demonstrate our technique using motion capture data from a sports video game. Our animated football player reacts to being hit with a ball, and then smoothly returns to the original motion sequence.

5.1.1 Implementation Details

The implementation of our method consists of two main components. The first is a preprocessing stage, where we use inverse dynamics to estimate the feedforward torques from the motion capture data. The second component, which happens during the dynamic simulation, is the calculation of the actual muscle torques. The muscle torques are a combination of the precomputed feedforward torques, and feedback torques which depend on the difference between the trajectories of the rigid bodies in the motion capture and the trajectories in the dynamic simulation.

Extensions to the Simulator to Support Inverse Dynamics

As we explained in Section 2.5.3, the row space of the Jacobian $\mathcal{J}$ is the space of possible constraint forces. We now wish to introduce an analogous matrix $\mathcal{H}$ whose row space is the space of all possible "muscle forces" which the joints can apply to their neighboring bodies[1]. The rows of $\mathcal{H}$ correspond to equal and opposite torques applied at the joint. The muscle forces are given by

$$\mathbf{f}_m = \mathcal{H}^T \boldsymbol{\tau}, \tag{5.1}$$

where $\boldsymbol{\tau}$ is a vector of multipliers, like the Lagrange multipliers, which are the magnitudes associated with the rows of $\mathcal{H}$. Adding the muscle forces into our forward dynamics equation (Equation 2.32) gives

$$\begin{pmatrix} \mathcal{M} & \mathcal{J}^T \\ \mathcal{J} & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathbf{v}} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{ext} + \mathcal{H}^T \boldsymbol{\tau} \\ -\boldsymbol{k} \end{pmatrix}. \tag{5.2}$$

When doing inverse dynamics, the acceleration $\dot{\mathbf{v}}$ is known, and the multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\tau}$ are unknown. We rearrange Equation 5.2 to get

$$\begin{pmatrix} \mathcal{J}^T & \mathcal{H}^T \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\tau} \end{pmatrix} = \begin{pmatrix} \mathcal{M}\dot{\mathbf{v}} - \mathbf{f}_{ext} \end{pmatrix}. \tag{5.3}$$

Details of Matrices $\mathcal{J}$ and $\mathcal{H}$

In Section 4.4, we saw how to compute J for different kinds of joints. Knowing J, we can determine H by making a couple of assumptions:

- The muscle forces and constraint forces originate at the joint, and must therefore apply an equal and opposite force to the two bodies that are connected at

[1] $\mathcal{H}$ has a similar structure to $\mathcal{J}$, and we will use the same fonts as we use for $\mathcal{J}$ (see page 22) when talking about the submatrices of $\mathcal{H}$.

the joint. Mathematically, this means that if the coordinates of J and H are written with respect to a frame centered at the joint (call it frame j) the row space of $\begin{pmatrix} {}_j\mathsf{H} \\ {}_j\mathsf{J} \end{pmatrix}$ must be a subspace of the row space of

$$\begin{pmatrix} & & & & & & & & & & & & & & & & & & & & & \\ \mathbf{0} & \cdots & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & \mathbf{0} & \cdots & -1 & 0 & 0 & 0 & 0 & 0 & \cdots & \mathbf{0} \\ & & 0 & 1 & 0 & 0 & 0 & 0 & & & & 0 & -1 & 0 & 0 & 0 & 0 & & \\ & & 0 & 0 & 1 & 0 & 0 & 0 & & & & 0 & 0 & -1 & 0 & 0 & 0 & & \\ & & 0 & 0 & 0 & 1 & 0 & 0 & & & & 0 & 0 & 0 & -1 & 0 & 0 & & \\ & & 0 & 0 & 0 & 0 & 1 & 0 & & & & 0 & 0 & 0 & 0 & -1 & 0 & & \\ & & 0 & 0 & 0 & 0 & 0 & 1 & & & & 0 & 0 & 0 & 0 & 0 & -1 & & \end{pmatrix}. \tag{5.4}$$

(the $\mathbf{0}$'s above are 6×6 blocks of zeros corresponding to the bodies that are unaffected by the joint. The remaining two 6×6 blocks are $\begin{pmatrix} {}_j\mathbf{H}_a \\ {}_j\mathbf{J}_a \end{pmatrix}$ and $\begin{pmatrix} {}_j\mathbf{H}_b \\ {}_j\mathbf{J}_b \end{pmatrix}$, corresponding to the two bodies, A and B, that are constrained by the joint.)

- We assume that the muscle force basis vectors (the rows of H) always span all of the degrees of rotational freedom in the joint. Mathematically, this means that $\begin{pmatrix} {}_j\mathsf{H} \\ {}_j\mathsf{J} \end{pmatrix}$ has rank 6. For instance, in a hinge joint, there is 1 degree of rotational freedom, and five DOF are removed from the system. For a ball joint, there are 3 degrees of rotational freedom, and 3 DOF are removed.

Given these two assumptions, it follows that the row space of $\begin{pmatrix} {}_j\mathsf{H} \\ {}_j\mathsf{J} \end{pmatrix}$ is identically the row space of the matrix in Equation 5.4.

Like we did in Section 4.4.1, we must again transform the left and right halves of the above matrix (Equation 5.4) into frames A and B respectively, as these are

the frames in which our implementation records the velocity twists for the bodies. This transformation results in the following formulas for the J and H matrices.

$$\begin{pmatrix} {}_a\mathbf{H}_a \\ {}_a\mathbf{J}_a \end{pmatrix} = \begin{pmatrix} {}^a j_x & {}^a j_y & {}^a j_z & 0 & 0 & 0 \\ {}^a k_x & {}^a k_y & {}^a k_z & 0 & 0 & 0 \\ {}^a l_x & {}^a l_y & {}^a l_z & 0 & 0 & 0 \\ 0 & r_{a_z} & -r_{a_y} & 1 & 0 & 0 \\ -r_{a_z} & 0 & r_{a_x} & 0 & 1 & 0 \\ r_{a_y} & -r_{a_x} & 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} {}_b\mathbf{H}_b \\ {}_b\mathbf{J}_b \end{pmatrix} = \begin{pmatrix} -{}^b j_x & -{}^b j_y & -{}^b j_z & 0 & 0 & 0 \\ -{}^b k_x & -{}^b k_y & -{}^b k_z & 0 & 0 & 0 \\ -{}^b l_x & -{}^b l_y & -{}^b l_z & 0 & 0 & 0 \\ 0 & -r_{b_z} & r_{b_y} & -1 & 0 & 0 \\ r_{b_z} & 0 & -r_{b_x} & 0 & -1 & 0 \\ -r_{b_y} & r_{b_x} & 0 & 0 & 0 & -1 \end{pmatrix}. \tag{5.5}$$

In the above, the vectors $\boldsymbol{j}$, $\boldsymbol{k}$ and $\boldsymbol{l}$ are the rotation axes and the forbidden axes of the joint.

Inverse Dynamics Preprocessing

Before beginning our dynamic simulation, we estimate a set of feedforward muscle torque multipliers $\boldsymbol{\tau}$ for each time step. Given the mass matrix $\mathcal{M}$, the constraint Jacobian $\mathcal{J}$, the matrix $\mathcal{H}$, the external force vector $\mathbf{f}_{ext}$, and the acceleration $\boldsymbol{a}$, we can calculate $\boldsymbol{\tau}$ using Equation 5.3.

We can estimate the accelerations of the rigid bodies in each frame by fitting a smooth curve to the position data, and then finding the second derivative of the curve.

One difficulty in evaluating Equation 5.3 is that the mass properties of the character's component rigid bodies are unknown. We deal with this by approximating the shape of the character with polyhedra and computing the mass matrix for these, assuming the density of water (the body's average density is reasonably close

to that of water).

We believe it may be possible to directly estimate the mass-inertia matrix of the figure from the motion capture data by making the mass properties unknowns in the inverse dynamics equation. However, this approach would require solving the inverse dynamics for every frame in the animation simultaneously, making it more computationally challenging. Estimating the mass properties from the geometry, on the other hand, allows us to solve each frame's inverse dynamics separately.

We can use Equation 5.5 to compute $\mathcal{J}$ and $\mathcal{H}$. To do this, we require a kinematic model of the character to tell us the location and type of each joint. In our experiments, this model was given with the data set.

Using Equation 5.3, we precompute feedforward torques for each of the frames in the animation before beginning the forward simulation. The total feedforward torque is given by $\mathcal{H}^T\boldsymbol{\tau}$. But for the next section, we will need to break this down into smaller components $\boldsymbol{\psi}_1, ..., \boldsymbol{\psi}_n$, each of which correspond to one degree of freedom of the joints.

$$\begin{aligned}\mathcal{H}^T\boldsymbol{\tau} &= \begin{pmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \dots & \boldsymbol{h}_n \end{pmatrix} \begin{pmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_n \end{pmatrix}, \\ &= \boldsymbol{h_1}\tau_1 + \boldsymbol{h_2}\tau_2 + \ldots + \boldsymbol{h_n}\tau_n, \\ &= \boldsymbol{\psi}_1 + \boldsymbol{\psi}_2 + \ldots + \boldsymbol{\psi}_n.\end{aligned} \tag{5.6}$$

The $\boldsymbol{\psi}$'s are stored in body relative coordinates rather than world coordinates, because the orientation of the joints with respect to the world may be different in forward simulation than in the original motion capture animation.

Combining Feedback and Feedforward Torques During Forward Simulation

The muscle torques applied during forward simulation are a combination of the feedforward torque (the torques calculated during the initial inverse dynamics phase) and the feedback torque, which models the muscle dynamics module of our motor control model.

Individual feedback torques are calculated for each degree of freedom of all of the joints. Let $\theta_1, \theta_2, ...\theta_n$ be joint angles corresponding to each degree of freedom, and $\dot{\theta_1}, \dot{\theta_2}, ...\dot{\theta_n}$ be joint velocities. The joint angles $\theta_{d1}, \theta_{d2}, ...\theta_{dn}$ are the "desired" joint angles – the joint angles from the motion capture data.

The feedback torque tries to compensate small drifts and disturbances during the simulation. It is given by

$$\gamma_i = \boldsymbol{h}_i|\tau_i|\Big(k_s(\theta_{di} - \theta_i) + k_d(\dot{\theta}_{di} - \dot{\theta}_i)\Big), \tag{5.7}$$

where k_s and k_d are the stiffness and damping constants. Note that the stiffness is proportional to the magnitude of the muscle torque, $\boldsymbol{h}_i|\tau_i|$, as observed empirically in muscle biomechanics.

Our total muscle torques are given by the sum of the feedforward and feedback torques:

$$\boldsymbol{\xi} = \sum_{i=1}^{n}(\gamma_i + \boldsymbol{\psi}_i). \tag{5.8}$$

The muscle torques are added into the dynamics equation along with any other external forces, such as gravity and perturbations.

Algorithm Summary

Our approach can be summarized by the following steps:

- Preprocessing: inverse dynamics
  - Estimate the mass matrix $\mathcal{M}$ for rigid bodies which approximate the shape of the character.
  - Fit a smooth curve to the position data.
  - Sample the accelerations of the rigid bodies at the rate at which we wish to run the dynamic simulation.
  - For each sampled time step:
    * Compute $\mathcal{J}$ and $\mathcal{H}$ given the positions of the rigid bodies.
    * Solve the inverse dynamics equation to determine the muscle torque multipliers $\boldsymbol{\tau}$.
    * Store the feedforward torques $\boldsymbol{\psi}_1, ..., \boldsymbol{\psi}_n$ in body coordinates.
    * Store the current joint angles $\boldsymbol{\theta}_d$ and joint velocities $\dot{\boldsymbol{\theta}}_d$.
- For each step during forward simulation:
  - Compute the current joint angles $\boldsymbol{\theta}$ and joint velocities $\dot{\boldsymbol{\theta}}$.
  - Compute the feedback torques $\boldsymbol{\gamma}_1, ..., \boldsymbol{\gamma}_n$, using Equation 5.7.
  - Compute the total external force $\mathbf{f}_{ext}$.
  - Solve the dynamics Equation 2.55 to determine the state of the system at the next time step.

5.1.2 Experiments and Discussion

We performed our experiments on captured arm motions and full-body motions in football games. Figure 5.1 shows the skeleton model and the surface model we use in our simulation and rendering. Although our skeleton model is a relatively complex dynamic system, some of the joints are highly simplified compared to their real-world counterparts. For example, the spine is modelled with only 3 segments, which will cause unrealistic body response under some cases. The true human shoulder and wrist joints are also much more complex than the simplified ones in our model.

Figure 5.2 shows motion perturbation on a sequence of arm motion. Figure 5.3 shows motion perturbation on a sequence of full-body motion. In both cases, the simulated skeleton responds to external disturbances and restores to the original motion naturally.

There are several limitations in this work. First, our motor control model is very simple right now. It can only cope with small disturbances. We work under the assumption that small disturbances are totally recoverable by muscles, without triggering the brain to replan the motion. In a real-life interactive sports video game, realistic dodges and realistic falls are absolutely desirable but very difficult. We know of no system that can do this yet. More sophisticated motor control models need to be developed.

We also simplify the system greatly by constraining the root joint (the joint located roughly at the Lumbosacral angle of the spine) to move along the motion capture path. We can thus omit the contact dynamics with the floor. This turns out acceptable since we are dealing with small upper limb perturbations. Even though motion perturbation is a subset of possible motion modifications, it is a large and common subset important for video games.

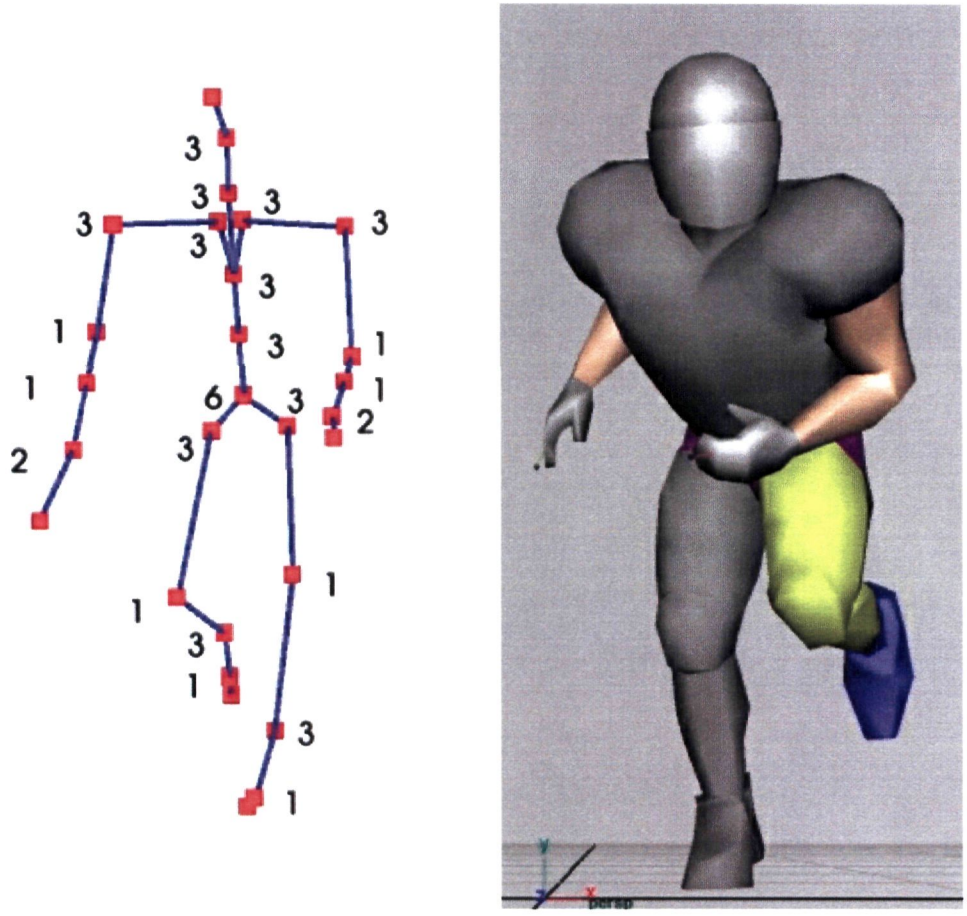

Figure 5.1: The skeleton model for our character simulation is shown on the left. Each square represents a joint. The total degrees of freedom is 54, and the degrees of freedom for each joint are labelled nearby. The skin geometry model for final rendering is shown on the right.

One thing we discovered while implementing this was the need for implicit integration methods (see Section 2.7.4). Without this extension, we were unable to set the feedback term in our motor control to sufficiently high values without causing troublesome numerical problems (which cause the links of the character's body to go flying off in wildly different directions).

Simulating realistic ground contact poses interesting questions for future

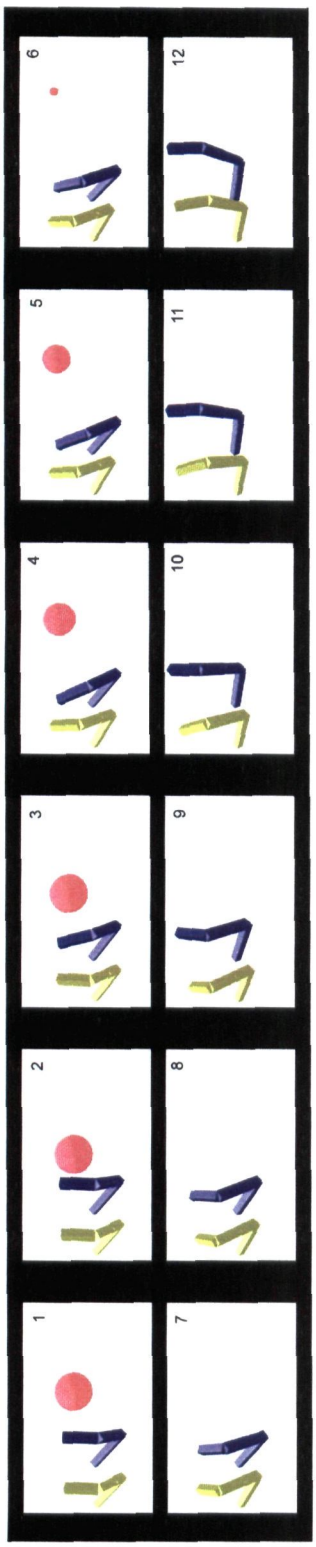

Figure 5.2: A sequence of a ball hitting a simulated arm (left to right, top to bottom). The motion capture data (left arm in each frame) is shown for comparison. The simulated arm reacts to the impact of the ball (which happens between frames 2 and 3), and then returns to the same path as the motion capture.

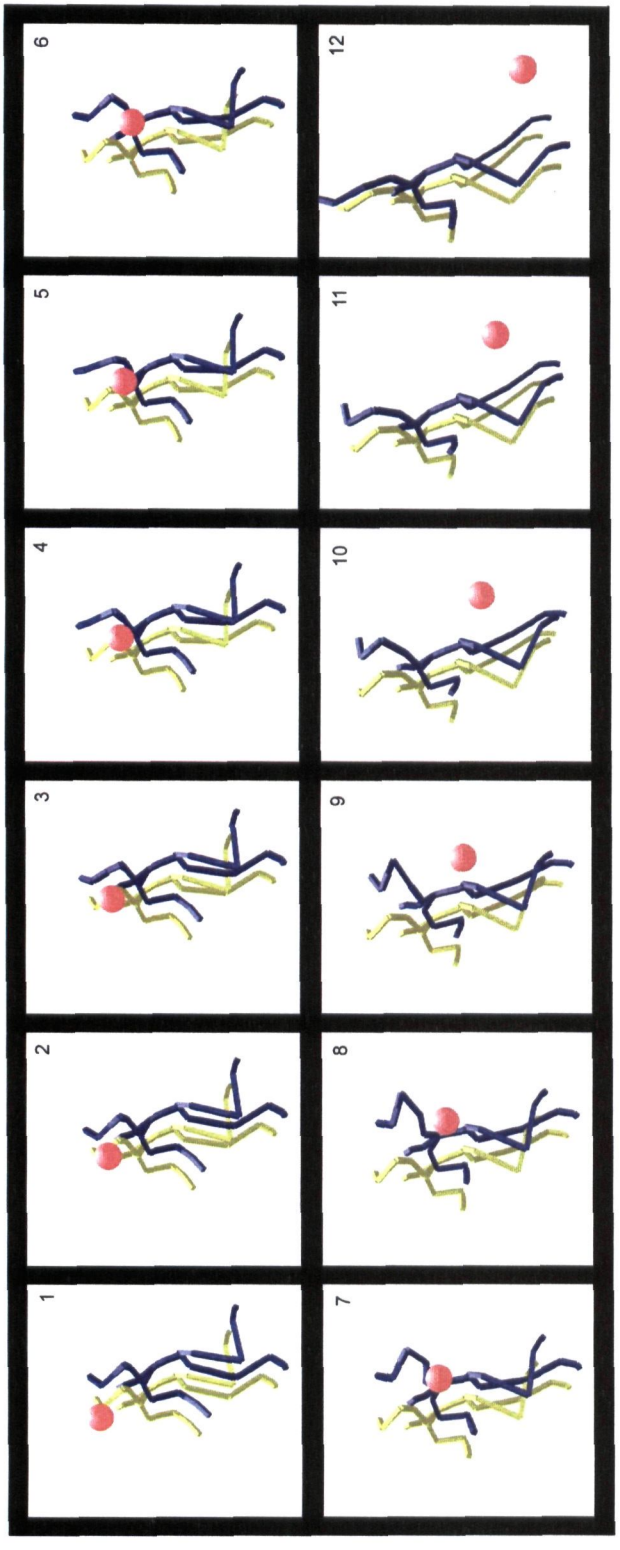

Figure 5.3: A sequence of a ball hitting a simulated body (left to right, top to bottom). The motion capture data (left body in each frame) is shown for comparison. The simulated body reacts to the impact of the ball (two collisions occur, one between frames 2 and 3, the other between frames 5 and 6), and then restores to the original motion.

work. Due to the statically unstable nature of human walking, the character would not be able to stay upright if we simply added ground contacts and removed the root joint constraint. One difficulty is the lack of a good foot model. Humans can carefully adjust the distribution of contact forces between the foot and the ground; something our simplified rigid body model could not capture. Even with a more accurate foot model it would probably be necessary to alter the foot placements slightly to maintain the balance, using a controller like that of Pai [Pai91]. It is unclear how to blend in the results of this kind of balance controller in a way that retains the essential qualities of the original motion capture animation. This is an area for future work.

5.2 Experiments to Test The Stabilization Algorithm

5.2.1 6-Link Chain

In the following test of our stabilization method, we simulate a 6 link chain falling freely under gravity (see figure 5.4). At time 0, the chain is completely horizontal. We observe the change in constraint error over time using one of three test conditions for comparison: no stabilization, Baumgarte stabilization, or post-stabilization. We evaluate the constraint function $\boldsymbol{g}(\mathbf{p})$ for each constraint, and use the maximum absolute value as a measure of constraint error. This number is roughly the largest joint separation distance. For comparison, each link in the chain is 100mm long. The time step size is 0.001 seconds.

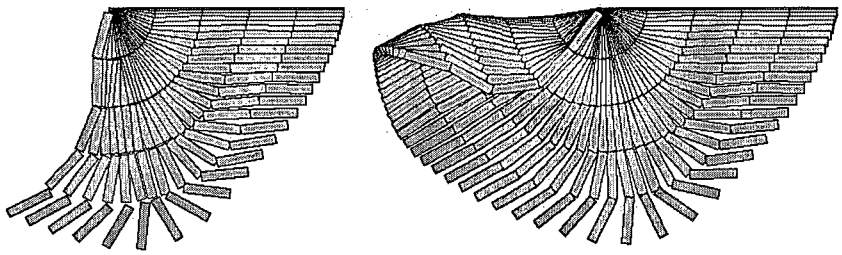

Figure 5.4: A simulation of a 6-link chain falling freely under gravity. Screen clearing between frames is turned off to show motion over time. In the version on the left, there is no constraint stabilization, so error in the constraints is evident as time progresses (note separation between the lowest two links of the chain). On the right, constraint stabilization keeps the links of the chain from drifting apart.

Figure 5.6 shows us how the simulation behaves in the absence of any stabilization. The error grows over time as the joints of the chain separate. The graph has stair steps because of the periodic nature of the swinging chain, which behaves something like a pendulum. The error grows rapidly when the chain is moving more quickly. By the end of 600 time steps (0.6 seconds), the error has climbed to around

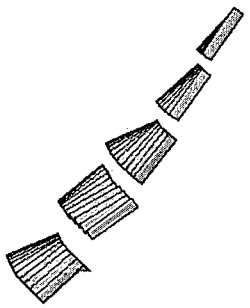

Figure 5.5: Unstabilized chain simulation: constraint error has grown very large. Links of chain no longer appear to be connected.

20mm.

In figures 5.7 and 5.8, we see the effect of Baumgarte stabilization on the error. The choice of constant used in Baumgarte stabilization has a large effect on how well the stabilization works. In figure 5.7, we show two choices of constant (50 and 200) that seemed to work best for this situation. Below 50, the error became much larger. With constant of 50, the maximum error is around 5mm, and when the constant is 200, we get slightly better results, with the error never exceeding 3mm.

If the constant is raised higher, the error does not continue to go down. Instead, we start to notice another problem: the chain starts to jiggle unrealistically. This corresponds to the error correction term growing so large that it overshoots its goal in a single time step. Figure 5.8 shows an example of this, using Baumgarte stabilization with a constant of 2000. When one watches this simulation, one notices the chain bouncing around rapidly due to the large stabilization forces. Correspondingly, the error graph is quite jagged, although quantitatively the error is not much worse than when a constant of 50 is used.

We show the results of using post-stabilization in figures 5.9 and 5.10. Figure

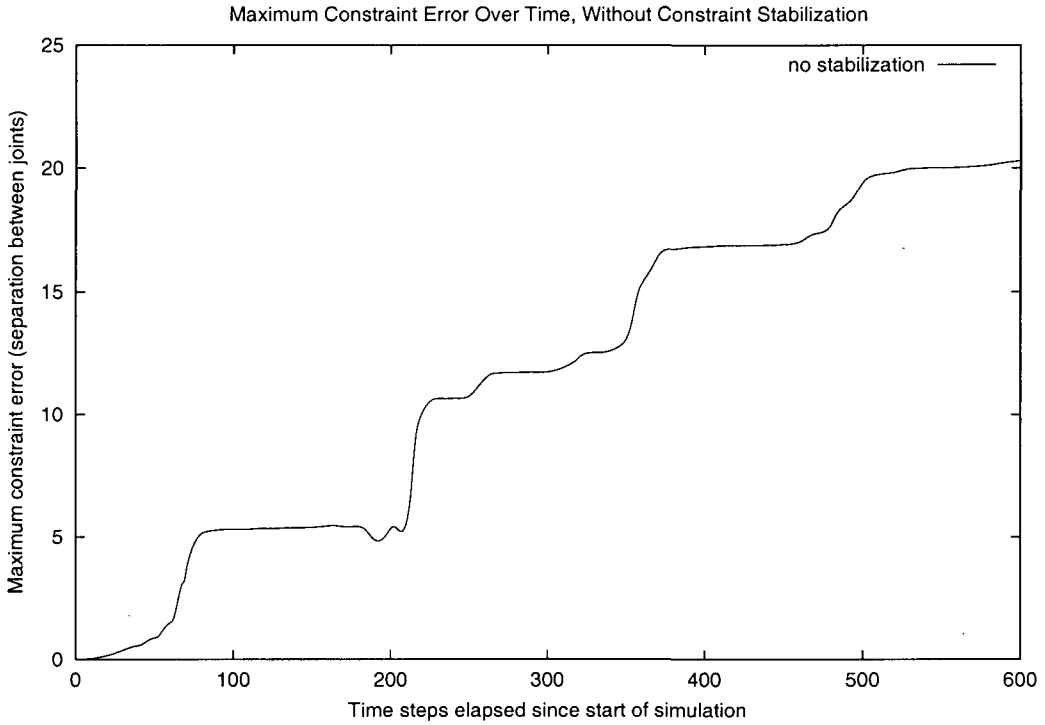

Figure 5.6: Maximum constraint error over time, for a simulation of a swinging 6-link chain without any stabilization

5.9 shows two curves: one is the constraint error measured in each step *before* the postStabilization step is applied, and the other is the error *after* the postStabilization step. Note that the scale in figure 5.9 is only one tenth that of the scale in the Baumgarte stabilization graphs above. Even in this scale, the "after postStabilization" curve is barely perceptible. Figure 5.10 shows just this curve, on an even smaller scale. The constraint error in this curve never goes above 0.01mm.

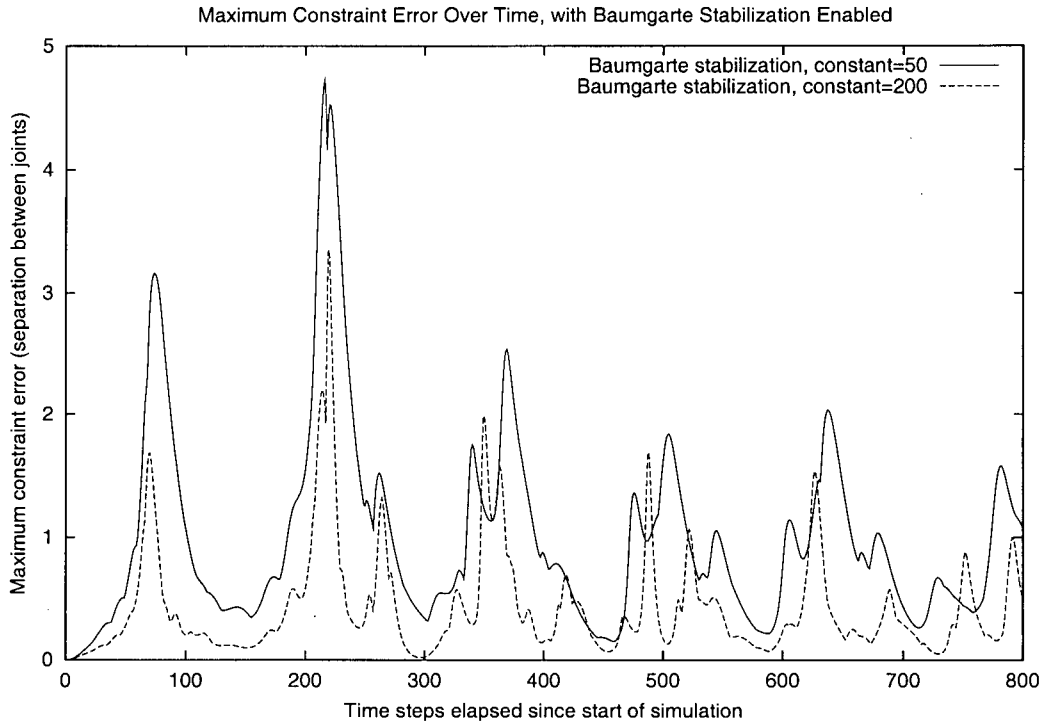

Figure 5.7: Maximum constraint error over time, for a simulation of a swinging 6-link chain with Baumgarte stabilization, using a constant of 50 or 200

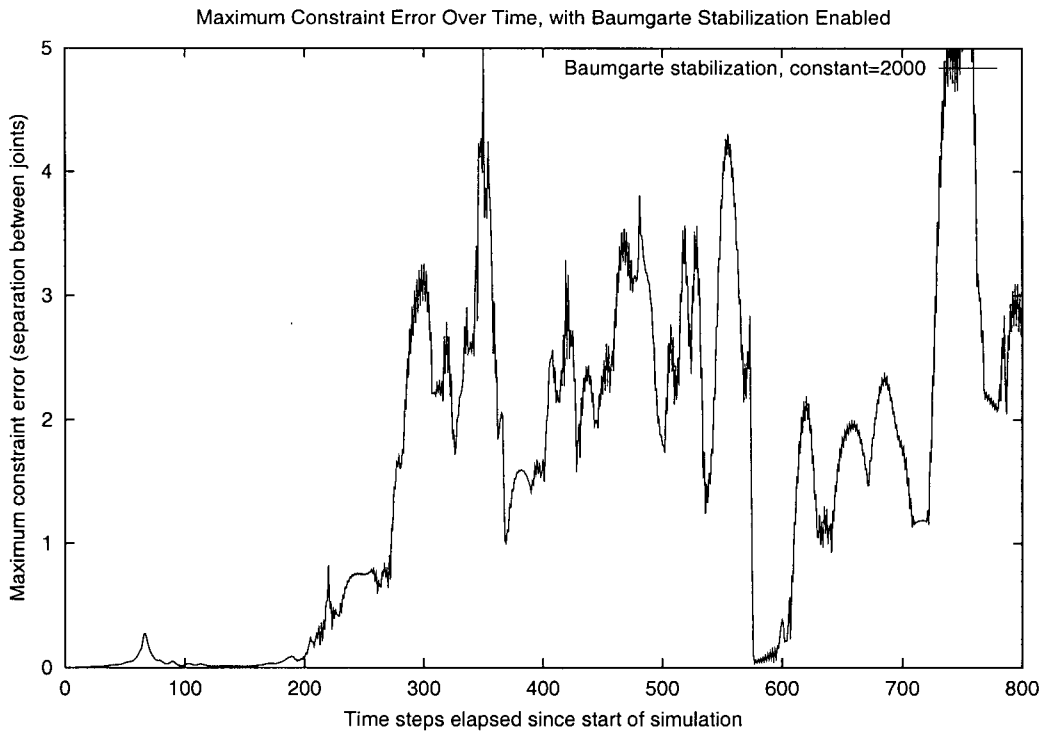

Figure 5.8: Maximum constraint error over time, for a simulation of a swinging 6-link chain with Baumgarte stabilization, using a constant of 2000

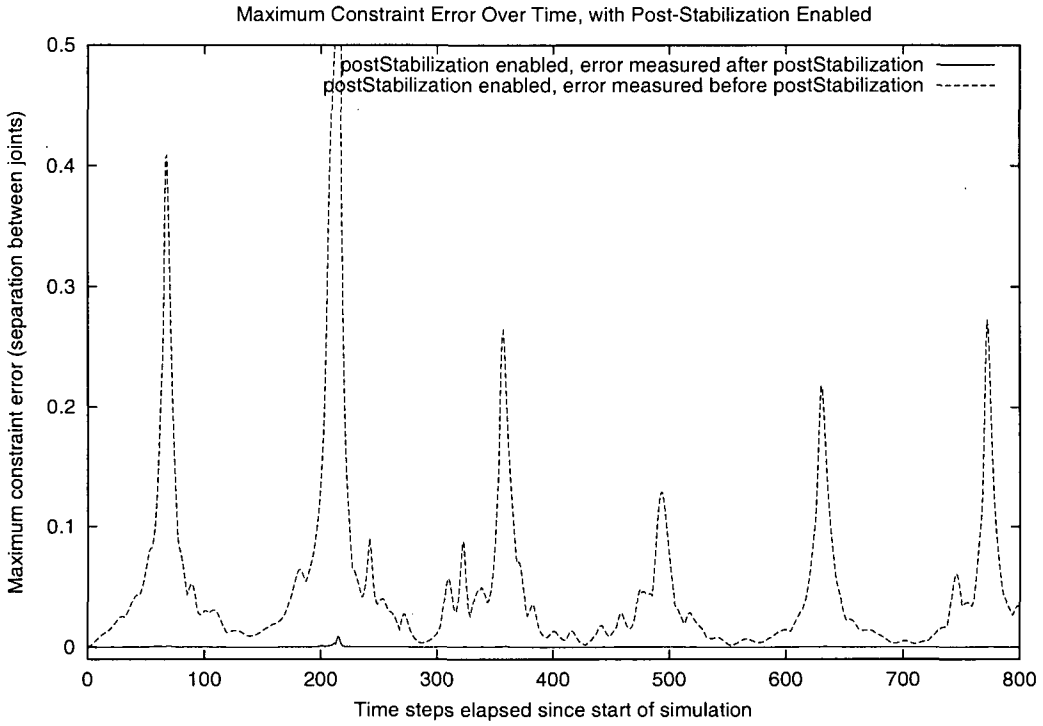

Figure 5.9: Maximum constraint error over time, for a simulation of a swinging 6-link chain with post-stabilization enabled. Error is shown both before and after applying the post-stabilization step.

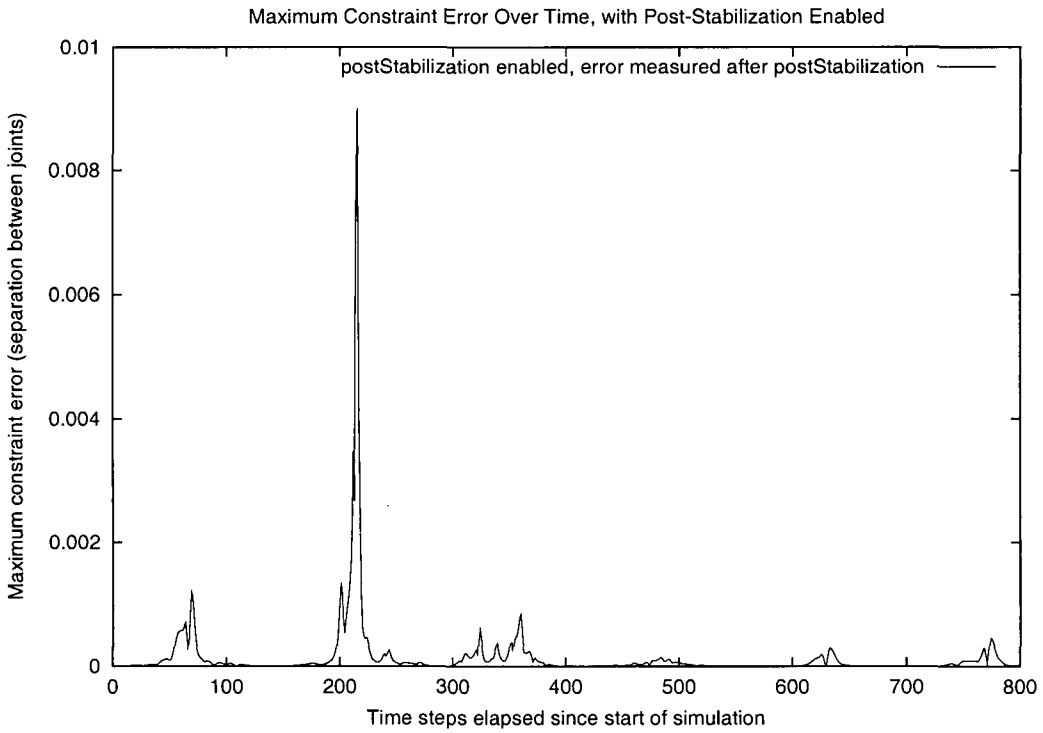

Figure 5.10: Maximum constraint error over time, for a simulation of a swinging 6-link chain with post-stabilization enabled. This graph is a close-up of the smaller curve in figure 5.9

5.2.2 Stabilization of Contact Constraints

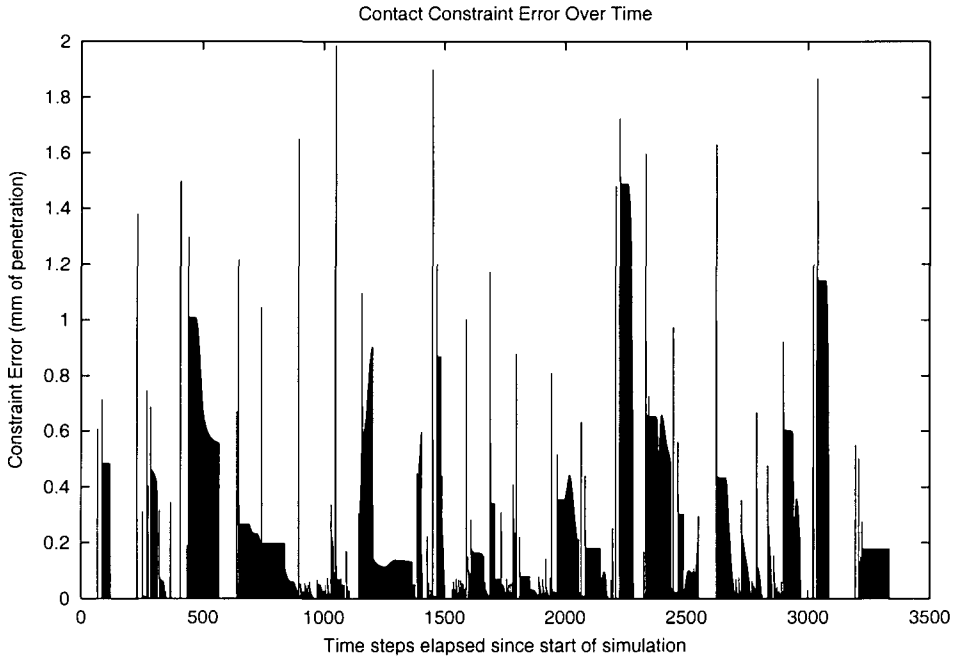

Figure 5.11: Contact constraint error over time for a rigid rectangle undergoing user interaction. No stabilization is used.

Contact constraints also have problems with constraint error. A contact constraint is said to have error if the separation distance at the contact is less than the *contact tolerance*. Figure 5.11 shows contact constraint error over a period of time when a rigid rectangle is pushed around on the screen by the user. Over this time period, the body experiences many different kinds of contact: impacts, resting contacts, sliding contacts and rolling contacts, sometimes with just one point of contact, sometimes with more than one (see figure 5.12 for some examples of typical motions).

When we run the same simulation with post-stabilization enabled, the constraint errors are eliminated completely. The post-stabilization scales well to simu-

Figure 5.12: Two screen captures from a contact simulation with a single moving rigid rectangle. User is pushing the rectangle interactively using repulsive forces at the mouse cursor.

lations with many bodies and many contacts, such as the simulation shown in figure 5.13.

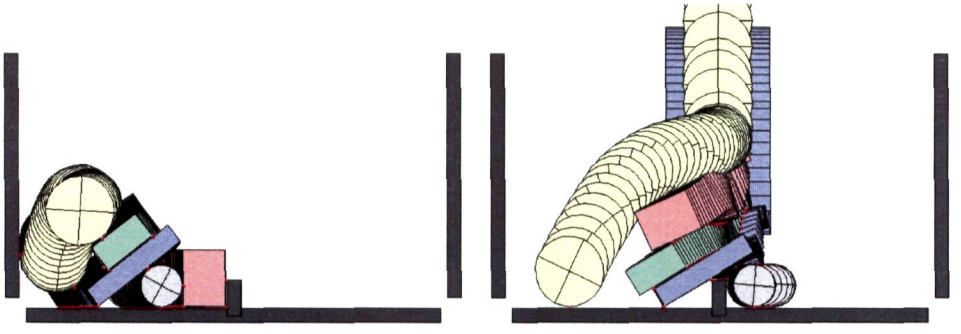

Figure 5.13: Two screen captures from a contact simulation with five moving rigid bodies.

5.2.3 Advantages and Disadvantages of Post-Stabilization

Comparing Baumgarte stabilization with our post-stabilization method, here are some of the trade-offs:

- Baumgarte stabilization uses special constants that must be carefully set by

hand, and often there is no perfect value for. Post-stabilization does not require any tweaking.

- With post-stabilization, the constraint error usually is eliminated in the same time step that it is created. With Baumgarte stabilization, usually it takes several time steps for the error to dissipate completely. The amount of time it takes to absorb the constraint error depends on the values for the special constants. If the constants are too small, then the error will dissipate slowly. If the constants are too large, the stabilization will over-correct, causing jerky oscillations of the bodies.

- Baumgarte stabilization is easier to implement, and involves very little extra computation per time step. It only involves measuring the constraint error and adjusting the constant $\boldsymbol{c}$ in the constraint equation $\mathcal{J}\mathbf{v} + \boldsymbol{c} = 0$. Post-stabilization actually requires formulating and solving another mixed LCP in addition to the dynamics LCP. This is quite a large penalty because solving these LCPs is one of the main bottlenecks in simulation performance.

- Post-stabilization can perform poorly when there are large errors in the constraints. This is because the linear approximation made in Equation 3.2 becomes a poor approximation. Usually this results in a disturbing crash of the simulation, with the bodies flying off wildly. It is also possible for the bodies to jump into interpenetration during the post-step in some circumstances. This can be helped some by restricting the size of the post-step, and taking multiple post-steps in the same time step, but with the penalties of having to solve more systems, and having to choose thresholds. In our experience, Baumgarte stabilization seems to perform better when there is a lot of constraint error,

especially if the constants used are not too large.

Chapter 6

Conclusions and Future Work

We have presented techniques for constructing a rigid body simulation system. We have described our implementation of a general-purpose system that can be used for interactive, real-time simulation. The system is a constraint-based system using Lagrange multipliers, and it incorporates a time-stepping method that handles multiple contacts with static Coulomb friction in a general way.

We presented a new constraint stabilization approach that combines the post-stabilization method of Ascher and Chin with an LCP formulation. This method is unique because it allows us to efficiently solve for the post-step when there are redundant contact constraints. Our experiments show that the post-stabilization eliminates constraint error very quickly.

We have shown that our software system can be extended and used as a tool to support other interesting research. Our example of this is a project involving human character simulation. We demonstrated a technique for using motion capture data to drive a rigid body simulation of a human, which can then react to small perturbations such as being hit by a ball.

6.1 Future Work

There are several directions in which this work could be built upon:

- *Hybrid system combining Featherstone's algorithm for large articulated bodies with Lagrange multipliers for contact.* Featherstone's algorithm, which uses reduced coordinates, is still faster for large articulated structures than the linear time Lagrange multiplier methods. Using reduced coordinates also carries the benefit of eliminating the need for constraint stabilization. Ideally, we would like a method where we could use an LCP framework that incorporates contact and friction, but also uses a reduced coordinate parameterization of the joints in the human kinematic model. The performance of the human character simulations in Chapter 5 would benefit greatly from such a method.

- *Real-time interaction through a data glove.* One area we would like to explore is more natural interfaces for interaction with the virtual world. Using a data glove is an obvious choice for a natural interaction method, but it is unclear exactly how to incorporate the kinematic data measured by the glove into a physical simulation. This problem is similar to the problem in Chapter 5 of taking kinematic motion capture data and using it to drive a physical simulation of a running character. Would a similar method work for real time interaction using a glove?

- *Improved collision detection performance.* Collision detection was not an area that we explored in much depth in this thesis. However, good collision detection routines are very important for interactive rigid body simulation. Some important issues are:

- *Exploiting temporal coherence.* Rigid bodies should only change position by a small amount in each time step. Collision detection algorithms should be able to exploit this property.
- *Graceful degradation under time constraints.* If we have hard time constraints (e.g., must maintain a certain frame rate), it is better for the collision detector to be able to return approximate results than no results at all. One way way to do this is to use a hierarchy of successively finer approximations to the real shape, such as sphere trees [Qui94]. O'Sullivan and Dingliana [OD99] present a method using sphere trees for collision response in interactive applications where only a fixed budget of time is available for collision detection and response.
- *Collision detector often returns too many contacts.* When the meshes are highly tesselated and we ask the collision detector to return all pairs of features that are within ϵ distance of each other, we may end up getting far more contacts than we really want. It would be ideal if the collision detector could have some smart way of returning just enough contact points so that there is no redundancy in the contact constraints.
- *Interpenetration Distance.* Interpenetration distance is more difficult to calculate than separation distance, Few collision detection packages allow interpenetration distance queries. However it is easier to make rigid body simulations robust when this information is available.

Many of these issues are being actively studied by collision detection researchers.

Bibliography

[ACR94] Uri M. Ascher, Hongsheng Chin, and Sebastian Reich. Stabilization of DAEs and invariant manifolds. *Numerische Mathematik*, 67(2):131–149, 1994.

[AG85] W. W. Armstrong and M. W. Green. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1:231–240, 1985.

[Ani97] Mihai Anitescu. *Modeling Rigid Multi Body Dynamics with Contact and Friction.* PhD thesis, University of Iowa, 1997.

[AP97] Mihai Anitescu and Florian Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14:231–247, 1997.

[AP98] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* Society for Industrial and Applied Mathematics, 1998.

[AP00] Mihai Anitescu and Florian A. Potra. A time-stepping method for stiff multibody dynamics with contact and friction. *Reports on computational mathematics, ANL/MCS-P884-0501, Mathematics and Computer Science division, Argonne National Laboratory*, 2000.

[APC97] U. Ascher, D. Pai, and B. Cloutier. Forward dynamics, elimination methods, and formulation stiffness in robot simulation. *International Journal of Robotics Research*, 16(6):749–758, 1997.

[Asc97] Uri M. Ascher. Stabilization of invariants of discretized differential systems. *Numerical Algorithms*, 14(1–3):1–24, 1997.

[Bar94] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of SIGGRAPH 1994*, volume 28, pages 23–34, 1994.

[Bar96] David Baraff. Linear-time dynamics using Lagrange multipliers. In *Proceedings of SIGGRAPH 1996*, volume 30, pages 137–146, 1996.

[Bar97] David Baraff. An introduction to physically based modelling: Unconstrained rigid body simulation. *SIGGRAPH '97 Course Notes*, 1997.

[Bau72] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1:1–16, 1972.

[BB88] R. Barzel and A. Barr. A modeling system based on dynamics consntraints. In John Dill, editor, *Proceedings of SIGGRAPH 1988*, volume 22, pages 179–188, 1988.

[BR79] O. Bottema and B. Roth. *Theoretical Kinematics*. North Holland Publishing Company, 1979.

[Can86] John Canny. Collision detection for moving polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2):pp. 200–209, 1986.

[CD68] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and Appl.*, 1, 1968.

[CPS92] R.W. Cottle, J.S. Pang, and R.E. Stone. *The Linear Complementarity Problem.* Academic Press, Inc., 1992.

[Cre89] J. F. Cremer. *An architecture for general purpose physical system simulation – Integrating geometry, dynamics, and control.* PhD thesis, Cornell University, 1989.

[CYP02] Michael Cline, KangKang Yin, and Dinesh Pai. Motion perturbation based on simple neuromotor control models. Technical Report TR-2002-03, University of British Columbia, 2002.

[Fea87] Roy Featherstone. *Robot Dynamics Algorithms.* Kluwer, 1987.

[GOM98] S. Gong, E. Ong, and S. McKenna. Learning to associate faces across views in vector space of similarities to prototypes. In *British Machine Vision Conference*, volume 1, pages 54–64, Southampton, UK, 1998.

[GvL89] G.H. Golub and C.F. van Loan. *Matrix Computations.* Johns Hopkins University Press, 1989.

[Hah88] James Hahn. Realistic animation of rigid bodies. In John Dill, editor, *Proceedings of SIGGRAPH 1988*, volume 22, pages 299–308, 1988.

[Hos] Wolfgang Hoschek. The colt distribution: Open source libraries for high performance scientific and technical computing in java. `http://tilde-hoschek.home.cern.ch/~hoschek/colt/index.htm`.

[KP02] P. G. Kry and D. K. Pai. Continuous contact simulation for smooth surfaces. *ACM Transactions on Graphics*, 2002. to appear.

[L̈82] P. Lötstedt. Mechanical systems of rigid bodies subject to unilateral constraints. *SIAM Journal on Applied Mathematics*, 42(2):281–296, 1982.

[LNPE92] C. Lubich, U. Nowak, U. Pohle, and Ch. Engstler. Mexx – numerical software for the integration of constrained mechanical multibody systems. Technical Report SC92-12, Konrad-Zuse-Zentrum fur Informationstechnik, 1992.

[Mas01] Matthew T. Mason. *Mechanics of Robotic Manipulation.* MIT Press, 2001.

[Mat] The matrix and quaternion faq, `http://www.cs.ualberta.ca/~andreas/math/matrfaq_latest.html`.

[MC95] Brian Mirtich and John F. Canny. Impulse-based simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*, pages 181–188, 217, 1995.

[Mir98] Brian Mirtich. Rigid body contact: Collision detection to force computation. Technical Report TR-98-01, Mitsubishi Electrical Research Laboratory, 1998.

[MLS94] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, 1994.

[Mur88] Katta G. Murty. *Linear Complementarity, Linear and Nonlinear Programming,*

http://ioe.engin.umich.edu/books/murty/linear_complementarity_webbook/. Internet Edition, 1988.

[MW88] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In John Dill, editor, *Proceedings of SIGGRAPH 1988*, volume 22, pages 289–298, 1988.

[OD99] C. O'Sullivan and J. Dingliana. Real-time collision detection and response using sphere-trees. In *15th Spring Conference on Computer Graphics*, pages 83–92, Budmerice, Slovakia, 1999.

[Pai91] Dinesh K. Pai. Least constraint: A framework for the control of complex mechanical systems. In *Proceedings of the American Control Conference*, pages pp.1615–1621, 1991.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *IEEE Intern. Conf. on Robotics and Automation*, pages 3324–3329. IEEE, 1994.

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. In B. A. Barsky, editor, *Proceedings of SIGGRAPH 1985*, volume 19, pages 245–254, 1985.

[ST96] D. E. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *Internat. J. Numer. Methods Engineering*, 1996.

[Str91] W. J. Stronge. Unraveling paradoxical theories for rigid body collisions. *Journal of Applied Mechanics*, 1991.

[Ull98] Chris Ullrich. Contact mechanics using green's functions for interactive simulated environments. Master's thesis, University of British Columbia, 1998.

[Ver74] A. F. Vereshchagin. Computer simulation of the dynamics of complicated mechanisms of robot manipulations. *Engineering Cybernetics*, 6:65–70, 1974.

[WGW90] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, pages 11–21, 1990.