Impulse-based Dynamic Simulation of Rigid Body Systems

by

Brian Vincent Mirtich

B.S.E. (Arizona State University) 1989
M.S. (University of California, Berkeley) 1992

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor John F. Canny, Chair
Professor David Forsyth
Professor Alan Weinstein

Fall 1996

The dissertation of Brian Vincent Mirtich is approved:

Chair Date

Date

Date

University of California at Berkeley

Fall 1996

Impulse-based Dynamic Simulation of Rigid Body Systems

by
Brian Vincent Mirtich

Abstract

Impulse-based Dynamic Simulation of Rigid Body Systems

by

Brian Vincent Mirtich
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor John F. Canny, Chair

Dynamic simulation is a powerful application of today's computers, with uses in fields ranging from engineering to animation to virtual reality. This thesis introduces a new paradigm for dynamic simulation, called *impulse-based* simulation. The paradigm is designed to meet the twin goals of physical accuracy and computational efficiency. Obtaining physically accurate results is often the whole reason for performing a simulation, however, in many applications, computational efficiency is equally important. Impulse-based simulation is designed to simulate moderately complex systems at interactive speeds. To achieve this performance, certain restrictions are made on the systems to be simulated. The strongest restriction is that they comprise only rigid bodies.

The hardest part of rigid body simulation is modeling the interactions that occur between bodies in contact. The most commonly used approaches are penalty methods, followed by analytic methods. Both of these approaches are constraint-based, meaning that constraint forces at the contact points are continually computed and applied to determine the accelerations of the bodies. Impulse-based simulation is a departure from these approaches, in that there are no explicit constraints to be maintained at contact points. Rather, all contact interactions between bodies are affected through collisions; rolling, sliding, resting, and colliding contact are all modeled in this way. The approach has several advantages, including simplicity, robustness, parallelizability, and an ability to efficiently simulate classes of systems that are difficult to simulate using constraint-based methods. The accuracy of impulse-based simulation has been experimentally tested and is sufficient for many applications.

The processing of collisions is a critical aspect of the impulse-based approach. Efficient algorithms are needed for detecting the large number of collisions that occur, without missing any. Furthermore, the physical accuracy of the simulator rests upon the accuracy of the collision response algorithms. This thesis describes these essential algorithms, and their underlying theory. It describes how the algorithms for simple rigid body simulation may be extended to systems of articulated rigid bodies. To prove the method is truly practical, the algorithms have been implemented in the prototype simulator, *Impulse*. Many experiments performed with *Impulse* are described.

Professor John F. Canny
Dissertation Committee Chair

Contents

List of Figures

List of Tables

Acknowledgments

This thesis would not have been possible without the help of many people. I thank my advisor, John Canny, who was an endless source of wisdom and inspiration. I thank the many great friends who made my seven years at Berkeley an experience to be treasured. Finally, I thank my family for their support, and especially Caroline, die mein Leben mit Freude erfüllt.

Chapter 1

Introduction

The field of classical dynamics is one of the success stories of modern mathematics. Dynamical systems exhibit a great deal of beautiful structure, and dynamicists have developed a body of knowledge that can predict the behavior of many of these systems to great accuracy. The ability to know in advance how our universe or some part of it will evolve is certainly one of the most powerful skills humans have learned. The modern computer, with its power to perform computations at blinding speed and display realistic images, has greatly increased our predictive power. But it has revealed holes in our knowledge as well: places where our predictive power breaks down. Predicting with certainty the outcome of a simple coin toss is at the limits of our current abilities, if not beyond them.

This thesis is about predicting the future, or more precisely, physical simulation. It deals with very simple systems and very simple models. The systems are comprised of rigid bodies; macroscopic deformations occurring in materials like cloth or rubber are not allowed. Different rigid bodies may however be connected by joints to form articulated bodies. Friction is modeled according to the Coulomb law; restitution is modeled using a single coefficient. The latter provides a crude way of accounting for microscopic deformation during a collision. Despite these simplifications, the predictive power is quite good for many applications. One common but significant simplification is also dropped: the qualification that one is only interested in the behavior of the system over some limited *modeling interval*. Most textbook problems are concerned with the behavior of a system over an interval where a single model applies: determine the velocity of the wrecking ball *before* it collides with the wall; or, determine the acceleration of the hoop down the ramp, assuming slipping does not occur. For the simulation problems addressed in this thesis, the analysis is not restricted in this manner. The simulator must produce correct motion for whatever situation and system the user describes, even in the presence of discontinuities produced by collisions, or changes in contact state.

The external influences that affect the motion of physical objects can be divided into two groups, based on whether or not they arise through physical contact. Forces due to gravity, external electric fields, or air resistance are examples of non-contact forces.[1] The motion of isolated rigid bodies, subject only to these non-contact forces, is well understood. For example, an accurate analytic model exists for the motion of a rigid body in a uniform gravitational field. The remaining influences on the motion of objects result from contact. The contacts between different objects are where all of the interesting behavior arises, and correct modeling of these contacts is critical to accurate physical simulation. Contact modeling is also where all of the difficult problems are. The models for contact are less accurate and more equivocal than those for the dynamics of isolated bodies. Much of the research in dynamic simulation has been devoted to contact modeling.

In studying contact interactions, there is the typical tradeoff between accuracy and efficiency. Finite element methods (FEMs), and related approaches, are among the most accurate methods for studying contact interactions, but they require significant computation time. They are not feasible, for example, in interactive simulation, where a user would like to watch a system evolve at a speed near the true speed at which it evolves in the physical world. This thesis is concerned with physically accurate simulation subject to the constraint of interactive or even real time performance. The accuracy of the methods is less than the accuracy of, for instance, finite element methods, but the goals are different. This thesis demonstrates that dynamic simulations can be computed at interactive speeds, and retain adequate physical realism and predictive power for real engineering applications. One of the fundamental assumptions made for computational efficiency is that all bodies are rigid.

Impulse-based simulation is a departure from the typical approach for performing interactive dynamic simulation. The greatest difference lies in how the contacts are modeled. Rather than computing explicit constraint forces to be applied at contact points, contact interactions are modeled exclusively through collision impulses applied between bodies. To understand why this approach might be useful, it is necessary to understand the standard constraint-based approaches, and their limitations.

[1]Air resistance might be modeled as a contact force resulting from collisions of a body with air particles. But it is more common to model it as an aggregate non-contact force based on the body's state.

1.1 Constraint-based contact models

Physically accurate, interactive simulation has received much attention in the fields of engineering and computer science. The standard approach uses a constraint-based model for contact, an outgrowth of the classical model for constrained dynamics. The latter is useful in computing the forward dynamics of articulated bodies: given the current positions and velocities of all parts of a physical system, find the current accelerations of all of the parts. Consider, for example, the planar double pendulum shown in Figure 1.1. Suppose

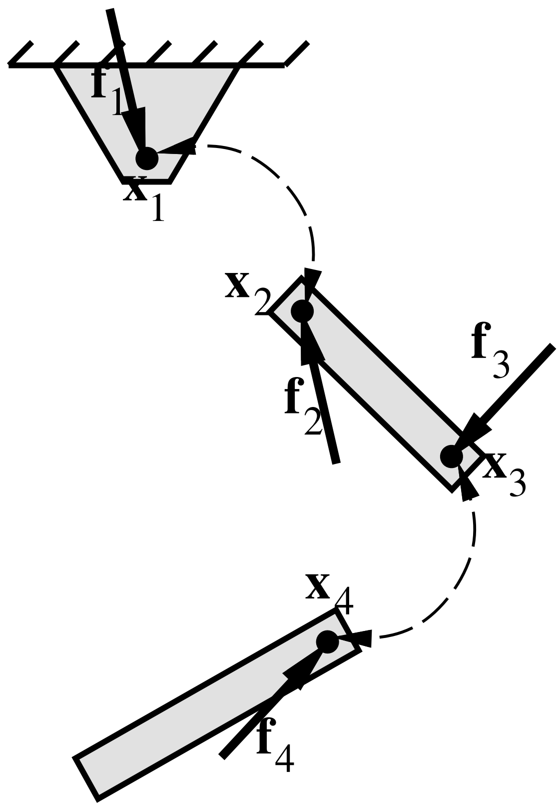

Figure 1.1: *A disassembled double pendulum. Although the pendulum is an articulated rigid body, its dynamics can be derived from the formulation for simple rigid bodies: first the constraint forces* $\mathbf{f}_i$ *are computed, and then the links are treated as individual rigid bodies.*

$\mathbf{x}_1$ and $\mathbf{x}_2$ are points that are on different bodies of the pendulum, but which are attached at a joint. However the two links move, the constraint

$$\mathbf{x}_1 - \mathbf{x}_2 = \mathbf{0} \tag{1.1}$$

must be enforced. This is done by introducing *constraint forces* that are transmitted through joints, maintaining the constraints. Once the constraint forces are known, the links of the system may be analyzed separately, and their motion determined from the Newton-Euler equations of rigid body motion. Some methods do not explicitly solve for the constraint forces, but produce the link accelerations directly. Usually the value of the constraint forces is not important, as long as the constraints are enforced, so these methods are suitable, and even preferable if computations can be reduced. General applications of the constraint-based approach to dynamic simulation and modeling are in [BB88, CS89, Gle94, IC87, WGW90].

1.1.1 Non-penetration contact forces

Sometimes bodies are not permanently attached to each other, but come into contact during the course of their motion. The row of dominos in Figure 1.2 is an example. These contacts give rise to *non-penetration constraints.* Unlike (1.1), a non-penetration

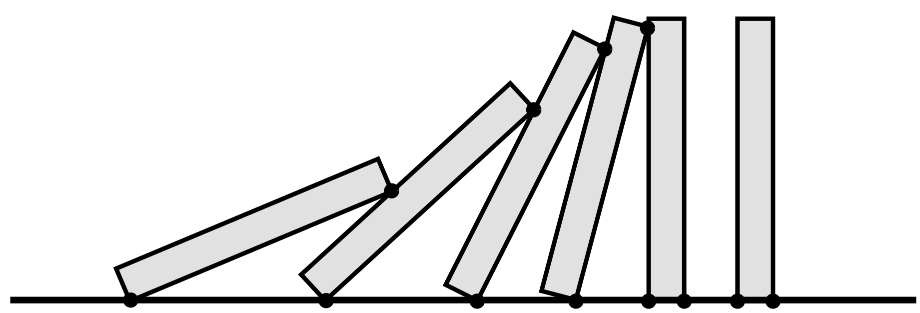

Figure 1.2: *For this system of dominos, there are no permanent constraints between bodies, but the contacts (indicated by dots) give rise to temporary non-penetration constraints between bodies.*

constraint is fundamentally an inequality, rather than an equality, that involves the accelerations of the bodies. In addition, there are restrictions on the constraint force: it can only act in a direction which tends to push the objects apart, not pull them together. If a pulling force is required to hold the objects in contact, the contact will merely break. For these reasons, non-penetration constraints can not be handled using the classical approaches for constrained dynamics.

There have been two main approaches to modeling non-penetration with constraints. The older and simpler approach is the family of *penalty methods.* These methods do not strictly enforce non-penetration; instead, they keep penetrations small enough so that they are not noticeable relative to the scale of the system. With this practical goal in mind, it is not surprising that the easy-to-implement penalty methods have dominated the computer animation field, where motion that "looks right" is often sufficient. Penalty methods vary in details, however the rough idea is the same. A stiff spring is attached at the contact, so that as two bodies move into one another, the spring attempts to push them apart. The more penetration, the stronger the restoring force. If the bodies move apart, the spring is destroyed. Penalty methods are used for rigid body simulation in [RH91, MW88, MZ90], and are also commonly employed in deformable body simulation.

Penalty methods have a few problems. The very large spring constants that are needed to keep the penetration sufficiently small generate large forces leading to stiff equations that are numerically intractable to integrate [WGW90]. Integration problems are exacerbated by the fact that springs can be created and destroyed in a very transient manner. Furthermore, choosing the spring constants is a black art. Constants that are satisfactory for one situation may be too high or too low for another, and the problem becomes worse when multiple bodies are in contact simultaneously. These spring constants do not correspond to familiar physical properties that can be measured, like the coefficient of friction. Another subtle problem of penalty methods arises because of the way collision detection is typically performed. Usually collision checks are done at discrete times rather than continuously over time intervals. As a result, contact is not detected until a finite amount of penetration has occurred. Under penalty methods, this causes forces that are larger than the true forces in the continuous system, causing instability.

Alternatives to the penalty methods are the analytic (or exact) methods for computing non-penetration contact forces, first studied by Lötstedt [Löt81]. The idea is to cast the non-penetration constraints as a linear complementarity problem (LCP):[2]

$$\begin{aligned} \mathbf{a} = \mathbf{A}\mathbf{f} - \mathbf{b} &\geq \mathbf{0} \\ \mathbf{f} &\geq \mathbf{0} \\ \mathbf{f}^T\mathbf{a} &= 0. \end{aligned} \tag{1.2}$$

Here, $\mathbf{a}$ is a vector of the relative normal accelerations at the contact points, and $\mathbf{f}$ is a vector of corresponding normal force components at these contacts. $\mathbf{A}$ and $\mathbf{b}$ are matrix and vector constants which are determined from the known configuration of the system.

[2]For the cursory treatment given here, Baraff's simpler notation is used [Bar94].

The constraint $\mathbf{a} > 0$ prevents motion that would cause penetration at the contact points. The constraint $\mathbf{f} > 0$ means that contact forces can only try to push bodies apart, not hold them together. The complementarity constraint $\mathbf{f}^T\mathbf{a} = 0$ means that at each contact point, either a force is acting and the relative acceleration is zero, or the bodies are separating, and the force is zero. The former case means the contact will persist beyond the current instant; the latter means the contact is ending. Lötstedt gives an algorithm for solving the LCP (1.2), based on the principal pivoting method of Cottle and Dantzig (see [CPS92] for a detailed treatment of LCPs). Lötstedt's algorithm in the context of rigid body simulation is described in [Löt84]. Baraff has also used LCP approaches for simulation of rigid body systems [Bar89, Bar92, Bar94].

When there is no friction in the system, the LCP (1.2) always has a solution for the contact forces, and this solution is unique, subject to certain non-degeneracy constraints [Bar92]. When friction is added to the model, the corresponding LCP may not have a solution, and if a solution exists it may not be unique. Lötstedt [Löt81] gives existence and uniqueness conditions, which Baraff [Bar92] generalizes. The most comprehensive results on the existence and uniqueness of solutions for rigid body contact forces, using the Coulomb friction model, are given by Pang and Trinkle [PT96] and Trinkle, *et. al.* [TPSL96]. Roughly speaking, existence and uniqueness can be guaranteed if the coefficients of friction are small enough. Certain problems, such as the static stability problem in which the bodies in the system are at rest, always have solutions.

When there is no friction, the LCP is convex and solutions can be computed using algorithms that run in worst case exponential time but expected polynomial time in the number of contacts. Convexity breaks down with the addition of friction. Baraff proved that the problem of determining the existence of a valid set of contact forces that obey Coulomb friction laws at the contacts is NP-hard when there is sliding friction. To avoid this problem, Baraff suggests a new model for consistency that includes non-colliding impulsive forces at some of the contacts. Under this model, he gives an algorithm to find a valid solution to the contact force problem. Its complexity is not known, although it appears to be practical in many examples [Bar91, Bar92]. None of the algorithms proposed for computing contact forces with friction are guaranteed to correctly terminate. Trinkle, *et. al.* give quantitative results using an approach based on Lemke's algorithm (this is also Baraff's approach), and also a feasible interior point method [TPSL96]. In general, the reliability of the algorithms decrease as the number of contact points increase.

Approximations are made to the Coulomb friction law in order to form an LCP for the contact forces. Under sticking conditions, Coulomb friction imposes the nonlinear restriction

$$\|\mathbf{f}_t\| \leq \mu \|\mathbf{f}_n\|,$$

where $\mathbf{f}_t$ and $\mathbf{f}_n$ are the tangential and normal forces at the collision point. To cast this into the LCP framework, a linear approximation to the friction cone is typically employed, such as a friction pyramid. This has the effect of making friction anisotropic for three-dimensional systems. Another common approximation is that upon the transition from sticking to sliding, the frictional forces are only partially (instead of directly) opposed to the tangential acceleration [Bar94, PT96].

Even without friction, analytic methods may not find the correct set of contact forces, due to contact degeneracy. Consider the idealized system of a table resting on a floor, where both objects are perfectly rigid, the table leg lengths are equal, and the floor is flat (Figure 1.3). Here, there are multiple sets of contact forces satisfying the constraints of the LCP: the forces are all positive, and prevent acceleration of the table into the floor. Without friction, the acceleration of the system is provably unique even when the contact

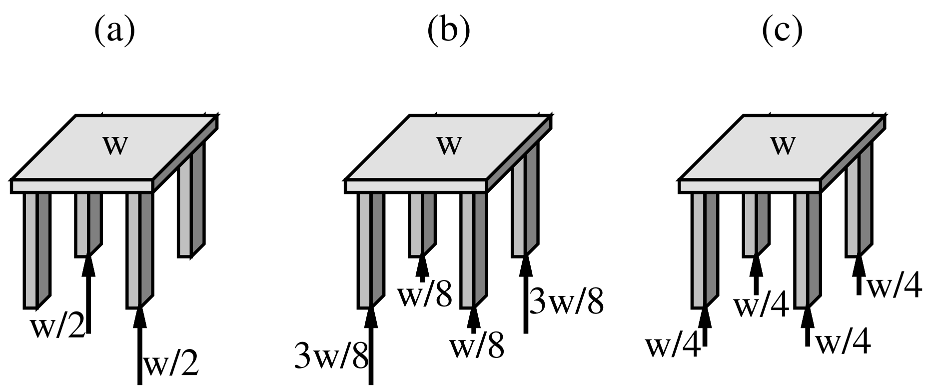

Figure 1.3: *For a symmetric table of weight w, there are infinite contact force configurations that satisfy the non-penetration LCP. Three are shown here.*

forces are not [Bar92]; in all three cases of Figure 1.3, the net acceleration of the table is $\mathbf{0}$. Analytic methods may not be reliable, however, for computing the contact forces. Force configuration (c) of Figure 1.3 is the correct one; scales placed under each leg of the table would all read the same weight. Analytic methods can compute different solutions, such as those shown in parts (a) and (b) of the figure, because the contact configuration is degenerate. On the other hand, penalty methods, and the impulse-based method described in the next section, always compute configuration (c). The idealized case is an important one to handle correctly; it is mathematically natural and the one often employed in simulations. It is an accurate approximation of certain real situations, for instance, when the table lengths are approximately equal, and the contacts are soft. This would be the case if the table leg tips are deformable, as on a folding table, or if the table rests on carpet or linoleum. If the table and floor are hardwood, and the leg lengths are different or the floor is warped, then the idealized model is not accurate, as the true forces on the legs may vary greatly. Such a system is not degenerate, and any of the simulation methods described will compute the correct contact forces.

With friction, there may be multiple contact force solutions that do *not* lead to the same accelerations. This indeterminacy that can arise in an analytic solution seems counter-intuitive to a deterministic view of the world. The inconsistency is even more disturbing: how can there be no "valid" solutions to the contact force problem for a physical system? Certainly, these forces always exist and are measurable for any physical system one can construct. These anomalies of the contact LCP reflect the limitations of the rigid body assumption and the Coulomb friction law, which are only approximations to reality.[3]

In dynamic simulation, exploiting coherence of the problem instances becomes critical for analytic methods. Solving a linear program is more difficult than solving a linear system of the same size, and solving LCPs is harder still. Solving a fresh linear complementarity problem at every time step would be prohibitively slow. Much effort can be saved by using the results from the previous time step. For example, in the pivoting algorithms of Lötstedt and Baraff, basic solutions to the LCP can be used from the previous time step, if they change relatively infrequently. Even more important, when friction is present there may be multiple solutions to the LCP. In order to preserve continuity of the accelerations of the bodies, the solution at the current time step must be computed with knowledge of the solution at the previous time step [Bar92]. Practically speaking, this dependence on contact coherence is perhaps the greatest limitation of the analytic methods.

[3]Tangentially related to this discussion is work of Peshkin and Sanderson [PS88], who have elegantly characterized the motion of parts sliding on a flat surface in the presence of Coulomb friction. Assuming nothing about the pressure distribution on the part, they compute an envelope of possible instantaneous centers of rotation, thereby bounding the possible motions of the part. Unfortunately, their method is based on a quasi-static assumption and the *minimum power principle* [PS89], which prevents application to many dynamic systems.

1.2 The impulse-based approach

Consider the vibrational part feeder of Figure 1.4. This machine shakes parts into

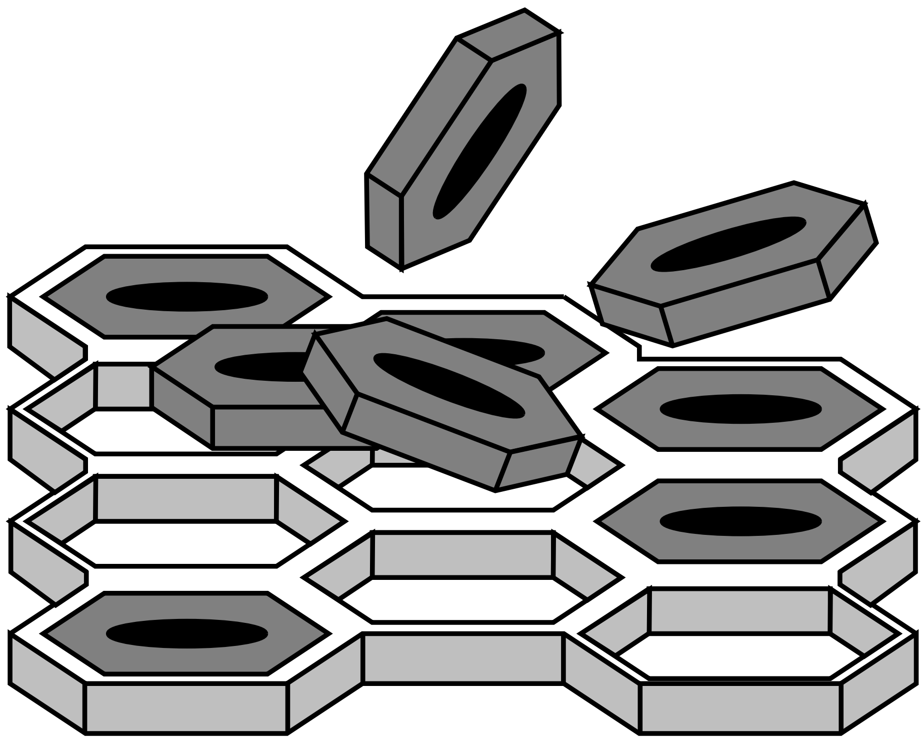

Figure 1.4: *A vibrational part feeder that shakes parts into recesses to be picked up by a manipulator. The system exhibits very transient contact modes.*

recesses so that they can be picked up by a manipulator. Simulating this device requires modeling the extremely transient contacts between the parts and the feeder. The contact interactions are dominated by collisions. The configurations of non-colliding contact exist only for short time intervals before changing. In short, the coherence between time steps that the analytic methods need for efficiency is destroyed. This part feeder example was one of the original examples that inspired the research described in this thesis. A method was desired for efficiently simulating these types of systems. The result is impulse-based simulation.

The fundamental idea of impulse-based simulation is that all contact between bodies is modeled through collisions at contact points. Non-penetration constraints do not exist; collisions are what enforce non-penetration between different bodies. Between collisions, the bodies move along ballistic trajectories, influenced only by gravity. As might be expected, the method adeptly handles systems like the parts feeder described above, where the bodies truly are unconstrained most of the time, and collisions are frequent. More surprising is the fact that impulse-based simulation can be used to model cases of continuous contact; it is a viable and powerful simulation paradigm for many applications. In the case of a book resting on a table, the corners of the book experience rapid collisions with the table, which prevent the bodies from penetrating. The rolling ball of Figure 1.5 rolls along the ground and up the ramp, becomes airborne, then bounces along the ground, eventually settling into a roll; sliding between the ball and terrain may also occur at various points during this evolution. Under an impulse-based model, all of these modes and transitions are handled by processing collisions. A macroscopic rolling constraint is never enforced. Rather, the macroscopic behavior results from treating the underlying collisions with a physically accurate model.

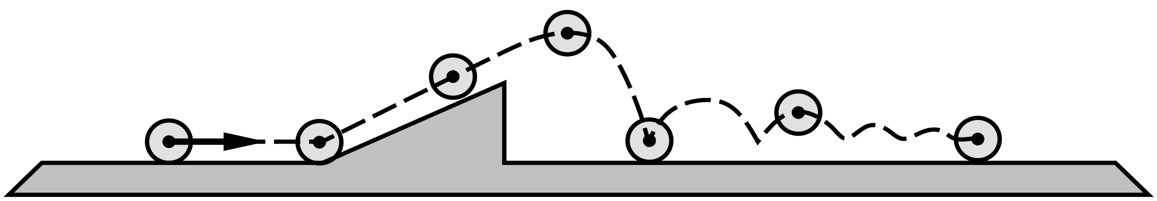

Figure 1.5: *Under impulse-based simulation, the various contact modes between the ball and terrain, and the transitions between them, are affected through a physically accurate model for collisions.*

Hahn pioneered this simulation paradigm. This thesis expands upon his work, and improves the methods for collision detection and collision response described in [Hah88]. The collision detection and response modules are critical components of an impulse-based simulator, due to the central role collisions play in the simulation. These modules must be very fast, due to the frequency of collisions, and also physically accurate, since collisions are the sole means by which contact forces are transmitted.

The top level impulse-based simulation loop is extremely simple. It comprises three steps:

1. **Collision detection.** Determine a maximum time interval over which the physical system may be integrated, such that no collision between two bodies will be missed. At the end of the interval, a pair of bodies called the *critical pair* must be checked for

collision.

2. **Dynamic evolution.** Evolve the system forward in time by integrating the equations of motion for each body. For rigid bodies, this trajectory is a simple ballistic one.

3. **Collision response.** Check to see if the critical pair of bodies has collided. If so, compute and apply a pair of collision impulses to them.

These three steps are repeated throughout the course of the simulation. The collision detection module works by maintaining an estimated time of impact for each pair of nearby bodies. This is only an estimate, and it is refined as the bodies move closer, but it is guaranteed to be conservative: the bodies can not collide any sooner than the computed time of impact. The integration step is determined by the earliest time of impact between any pair of bodies. The bodies that have the minimum time of impact become the critical pair.

Once the integration interval is determined, all of the bodies in the system can be evolved without regard to collisions, since none will occur during the interval. More precisely, the surfaces of any bodies that collide during the interval will still be touching (within some specified tolerance) at the end of the interval. The dynamic evolution step is very simple because the bodies are treated as unconstrained,[4] and so their motion is easy to compute. For example, isolated rigid bodies follow ballistic trajectories during the integration step. The dynamic integration step is highly parallelizable since the bodies are moving independently during the integration. In a computer with multiple processors, the bodies can be parceled among the processors, and the entire system can be integrated without any need for inter-processor communication. This can result in a substantial speedup for physical systems with many objects. It is an advantage over constraint-based methods, which need the state information of many bodies in order to compute the motion of each one.

If the critical pair of bodies are touching after the integration step, and have velocities that indicate a collision should occur at the contact point, the collision response module computes a pair of equal and opposite collision impulses to apply at the contact points. These impulses instantaneously change the velocities of the colliding bodies, send-

[4]This statement will be relaxed slightly in the context of hybrid simulation. But in both pure impulse-based simulation and hybrid simulation, there is no interaction between bodies through non-penetration constraints.

ing them along new trajectories. For physical realism, the collision response module should account for friction and energy loss during collision. Much of the work in rigid body simulation uses a simple model for collision resolution. The typical model assumes pre- and post-collision velocities can be related with algebraic equations, without regard to the underlying physical processes that occur during collision. Simple restitution laws that can add energy to the system are typically used. For impulse-based simulation, the collision response model is critical to the physical accuracy of the simulator, and better methods are needed. Collisions are resolved by integrating the dynamic state according to differential equations that hold *during* the colliding contact. These equations are derived directly from the underlying frictional and restitutional laws; they are more accurate than the algebraic approximations.

One might argue that impulse-based simulation is not a valid contact model. *A rock sitting on a table is not undergoing a series of collisions. It's just not moving, period.* This is certainly true, although all approaches to simulation make approximations and deviate from reality, especially those that achieve interactive simulation. The question is: what information is desired from the model? Summing the collision impulses delivered from the ground to the rock over some time interval, and dividing by the time interval, will yield the same average force delivered to the rock as a constraint-based approach would. The average velocity of the center of mass of the rock also vanishes, as in constraint-based methods. Finally, the piecewise ballistic trajectory of the rock results in little displacement of the rock from its average position. With a small enough collision tolerance, the displacement can be made less than a pixel, so that the motion of the rock on a graphics display is imperceptible. Although the impulse-based model deviates from reality, it preserves important physical quantities related to forces and the motion of bodies.

Based on the literature, the impulse-based method has been more carefully tested against reality than the constraint-based interactive simulation methods. Many comparisons of impulse-based simulation to physical experiments or theoretical predictions are given later in this thesis. More comparisons to physical experiments are certainly needed, for both simulation paradigms.

1.3 Impulses versus constraints

There are several advantages of the impulse-based approach:

- **No contact coherence required.** Analytic methods rely on contact coherence for interactive speed, and even more importantly, for evolving the system in a physically plausible way when multiple solutions exist to the contact LCP [Bar92]. Contact coherence is not required for impulse-based simulation; extremely transient contact modes cause no slowdown. This is especially useful in situations where objects settle as they make a transition to a resting state.

- **Computational robustness.** As with the penalty methods, nothing can go wrong in computing contact interactions. Given the dynamic states of the colliding bodies and the material dependent parameters, a unique impulse to resolve the collision can always be computed; the collision resolution algorithm always terminates with a valid solution. The NP-hard problem of computing (non-colliding) contact forces is avoided, as are the indeterminacy and inconsistency associated with analytic methods. The stiff differential equations of penalty methods are also avoided.

- **Colliding contact naturally handled.** With constraint-based methods, collisions must be handled separately, before any contact force analysis begins. But under impulse-based simulation, colliding contact is naturally handled, since collisions are used to model all contact.

- **Non-penetration strictly enforced.** Like analytic approaches but unlike penalty methods, impulse-based simulation strictly enforces non-penetration. In addition, if a heavy object and a light object are sitting on a table, both will experience collisions with the table at roughly the same frequency; the heavy object does not require more impulses.

- **Decoupling of bodies.** Unlike constraint-based approaches, bodies are evolved independently during dynamic integration under impulse-based simulation. This natural decoupling facilitates parallelization. Collisions are also processed using a completely local model, so the global state of a system of contacting bodies is not needed to compute the impulses. A description of preliminary work in parallel impulse-based simulation is in [Pau95].

- **Simple design.** The impulse-based paradigm is conceptually simpler than analytic methods, making it easy to code. Advanced numerical methods are not required. All macroscopic contact modes (rolling, sliding, resting, and colliding) are handled with a single, local collision response algorithm.

- **Physically valid contact forces.** Like penalty methods, the impulse-based method is not subject to the contact degeneracies that analytic methods are. In the example of Figure 1.3, the physically valid contact forces will be computed. (Forces are computed by time averaging impulses over a short interval.)

- **Extension to constrained bodies.** The methods of *pure* impulse-based simulation are easily extended to handle systems of rigid bodies linked together by joints.

Though not an inherent property of impulse-based simulation, the collision response model presented in this thesis is a more accurate model than those typically employed in interactive simulation. It would be useful in other simulation paradigms as well. There are also disadvantages to the impulse-based method as compared to constraint-based methods. In particular:

- **Poor handling of stable and simultaneous contact.** Impulse-based methods are less efficient than constraint-based methods for certain types of prolonged, stable or simultaneous contact; in some cases the impulse-based methods are not even feasible. The canonical example is a stack of blocks at rest.

- **Static friction creep.** Artifacts of the impulse-based approach appear when modeling static friction in certain cases. A block will creep down a ramp, even if static friction should hold it in place.

These limitations are significant, and this thesis does not claim that the impulse-based approach is a replacement for constraint-based approaches. There are cases where impulse-based simulation is the most natural approach for modeling non-penetration, and where it leads to faster simulation speeds (based on reported data). The same can be said of constraint-based methods. In the literature, it is evident that constraint-based approaches have been applied to problems they are well suited to, and the same is true of impulse-based approaches. Any contest between the two methods would be decided before it began by the choice of physical system to simulate. Furthermore, such a contest would be pointless, since the methods are complementary more than they are competing: the situations for which one approach works poorly are exactly those for which the other approach works well. Previous work has demonstrated what can be done with constraint-based techniques; less has appeared on what can not be done easily. One purpose of this thesis is to describe what can and can not be done with impulse-based techniques.

Much of this thesis deals with theoretical issues surrounding impulse-based simulation, however, there is a practical component as well. The ideas discussed in the thesis have been implemented in *Impulse*, a prototype impulse-based dynamic simulator. *Impulse* served as a proof by example during the research phase of this thesis; only through implementation could the techniques be proven practical. While many of the discussions refer to this specific implementation, they apply to any impulse-based simulator.

1.4 Overview of the thesis

The remainder of the thesis is organized as follows.

Chapter 2 describes collision detection in the context of impulse-based simulation, using the broad phase and narrow phase detection systems in *Impulse* as examples. In particular, the chapter discusses how collision checking may be performed conservatively, so that collisions are guaranteed to be detected.

Chapter 3 describes the collision response model used in *Impulse*. A physically accurate response model is critical to accurate impulse-based simulation. The model incorporates friction and restitution, and analyzes the dynamics that occur *during* a collision.

Chapter 4 derives Featherstone's algorithm for the forward dynamics of constrained bodies from the first principles of Newtonian physics. This is a key ingredient for the simulation of constrained bodies, and also serves as the basis for constrained body collision response.

Chapter 5 describes hybrid simulation, a method of applying impulse-based simulation to systems of constrained rigid bodies. It discusses necessary modifications to the collision detection and collision response algorithms described earlier, as well as a framework for supporting control and behavior systems.

Chapter 6 is self-contained, and presents algorithms for computing the center of mass and moments of inertia for arbitrary uniform-density polyhedral bodies. The algorithms are very fast and naturally minimize numerical errors; they are useful for any rigid body simulator.

Chapter 7 discusses many examples of simulations and experiments performed with *Impulse*, illustrating the strengths and weaknesses of the impulse-based approach. Accuracy is analyzed qualitatively and quantitatively, execution times are provided, and several real applications from the part-feeding domain are discussed.

Chapter 8 concludes the thesis and discusses future work.

Appendix A provides a variety of basic information relevant to dynamic simulation, including the Newton-Euler equations for rigid body motion, and how orientation is parameterized and evolved using a quaternion representation.

Chapter 2

Collision Detection

A fundamental problem in the simulation of physical systems is the collision detection problem: determining when and where two bodies come into contact. This problem has its roots in computational geometry and robotics. Typically, in the former setting, a group of static objects are tested for intersection, while in the latter setting, the time of contact must be reported between bodies following trajectories that are closed form functions of time. In simulation, the physical system is certainly not static, and the paths of the objects are described with differential equations instead of closed form trajectories. The speed requirements that interactive simulation demands are also formidable. Hahn found collision detection to be the bottleneck in simulation of physical systems [Hah88]; in his experiments, collision detection often accounted for over 95% of the computation time. Collision detection algorithms have improved since then, however, they remain the bottleneck in many situations.

This chapter describes how collision detection may be performed in an impulse-based simulator, using the system in *Impulse* as an example. The main contribution is an algorithm than can efficiently detect a large number of collisions, and that is guaranteed not to miss any collisions. To meet this guarantee, lower bounds on the time of impact of two ballistic rigid bodies are derived from the principles of dynamics. Methods of generalizing this approach to other types of motion are described. This chapter also develops a fast bounding box technique for culling most of the collision checks in a simulation with many bodies. Two variants of the algorithm are described and compared, and the algorithm is also compared to other bounding box techniques.

2.1 Introduction and related work

Hubbard gives a comprehensive reference list of work in collision detection [Hub94]. Early approaches to collision detection for simulation solved problem instances from scratch at every time step. An example is the polyhedral collision detection algorithm of Moore and Wilhelms that is $O(n^2)$, where n is the number of features on both polyhedra [MW88]. Hahn's method tests every edge of one polyhedron against every face of the other when bounding boxes can not guarantee the absence of collision, and therefore also has quadratic complexity [Hah88]. These types of algorithms are too slow for many simulation applications. Using bounding boxes or spheres helps, but is not enough. Gilbert, *et. al.* proposed a $O(n)$ algorithm for determining the distance between convex polyhedra, which also provides a measure of penetration when polyhedra overlap [GJK88]. The biggest improvement over these early algorithms came from the realization that coherence could be used to greatly reduce the computations. In simulation, the collision detection system solves a series of related problems, each one only slightly different than the one before. For convex polyhedra, coherence is combined with locality: using local properties to verify separation between bodies, or lack thereof. Gilbert, *et. al.* describe an adaptation of their algorithm to take advantage of coherence when it exists, however the reported speedups are fairly modest [GJK88]. Coherence is used more effectively in Baraff's witness plane algorithm [Bar92], and is thoroughly exploited in the Lin-Canny closest features algorithm [LC91, Lin93].

For simulation, the collision detection algorithm is only called at discrete sample points. Even if invocation occurs more than once per frame, it is still possible to miss collisions. One pathological example is a bullet speeding toward a thin wall; No matter what the minimum sampling period of the collision detection system (the *minimum temporal resolution* [Hub96]), one can choose a bullet speed and wall thickness such that the bullet passes completely through the wall between collision checks. One correct solution is to apply detection algorithms to the four-dimensional hyper-polyhedra swept out in space-time [Can84]. These methods are too slow and have not been used in any dynamic simulators described in the literature. Most simulators systems handle the problem of missing collisions by ignoring it [CS89, MW88, Hah88, Bar90].

Ignoring the problem is not a good solution, especially in an impulse-based paradigm, where collisions are used to model contact forces. Von Herzen, *et. al.* [HBZ90] present an algorithm that uses Lipschitz bounds to derive limits on how far parts of a parametric surfaces can move over a time interval; their system is guaranteed to catch all collisions. The bounds must be supplied by the user when the surface is defined. The algorithm of Snyder, *et. al.* uses an interval version of Newton's method for root finding to achieve the same goal: guaranteeing that the very next collision will be detected [SWF+93]. Here, Lipschitz bounds are not needed since the exact trajectories of the surfaces over time are input data for the problem. These approaches are similar to the one used in *Impulse* and discussed in this chapter. Lower bounds on the times of impact between bodies are maintained and used to ensure collisions are not missed. The user is not required to provide Lipschitz bounds nor the exact trajectories of the bodies over time. Physical laws, such as the conservation of momentum, provide enough information to bound the times of impact. Like the systems described in [HBZ90, SWF+93], *Impulse* uses a heap structure to schedule collision checks between bodies.

2.1.1 Collision detection in *Impulse*

For computational speed, *Impulse* restricts bodies in the simulation to be rigid and polyhedral. Rigidity allows much computation to be performed ahead of time, only once. The polyhedral restriction is is fairly mild one since any more general shape, such as a parametric surface or CSG-style solid, can be approximated to arbitrary closeness with a polyhedral model. Of course the complexity of the polyhedral model may grow very large, but the algorithms used by *Impulse* are quite insensitive to this complexity. *Impulse's* collision detection system is *exact*, meaning that the underlying polyhedral models of the objects are eventually used if the objects are close enough. These methods are in contrast to *approximate* methods, which are less concerned with locating the exact collision point and more concerned with performance. Hubbard's method based on bounding sphere hierarchies is one approximate method which is able to perform collision checking between very complex bodies at frame rates [Hub96]; Hubbard makes a good case for using approximate methods in many time-critical applications. Such approximate methods were not used in *Impulse*, which was designed to have value as a simulation tool. Inaccuracies in the position of contact points can have large repercussions in the course of the simulation.

Efficient collision detection systems employ some type of multi-level strategy. Bounding boxes, bounding spheres, octrees, or similar methods are used to prune most of the potential collision checks between a group of bodies. A more sophisticated but expensive collision detection algorithm is applied to pairs of bodies that can not be dismissed by the simple technique. This chapter uses Hubbard's terminology, referring to these two levels of checking as the *broad phase* and the *narrow phase*. For a simulation with n bodies, the broad phase prunes most of the $O(n^2)$ pairs, and the narrow phase is applied to the pairs that remain.

In the broad phase, *Impulse* computes bounding boxes for the swept volumes of bodies over a time interval, and finds intersections of these boxes using a hierarchical hash table. The swept volume technique is different than a four-dimensional space-time intersection test in two respects. First, the bounding boxes are only conservative estimates of the swept volumes of the bodies, with much simpler geometry than the true swept volumes. Second, since the boxes are not being used to compute the time of collision, but only to signal that one might have occurred during some time interval, one can project the true swept volume in space-time along the time axis, into the physical space. The analysis is simplified by working in three-dimensional physical space rather than in four-dimensional space-time. The hashing technique employed for detecting bounding box overlaps is based on a strategy Overmars proposes for solving the static point location problem [Ove92].

Impulse's narrow phase collision detection is based on the Lin-Canny algorithm. It is one of the fastest algorithms known for tracking the closest features between convex polyhedra, in a setting where coherence can be exploited. The algorithm's output can easily be used to compute the distance between the polyhedra, which serves as a basis for collision detection. The very code which implements the Lin-Canny algorithm in *Impulse* was subsequently used for the low-level core of the *I-COLLIDE* collision detection package for large-scale environments [CLMP95].

Both the broad and narrow phases of *Impulse's* collision detection system rely heavily on dynamics as well. Dynamics are used to compute bounding boxes that enclose bodies' swept volumes during the broad phase. Dynamics are used to determine when the next collision check should be performed between a pair of objects during the narrow phase, in a way that guarantees collisions will not be missed. This coupling between collision detection and dynamics is unfortunate from a modularity standpoint. Collision detection can not be isolated as a strict geometric problem; the detection system must have information about the dynamics of the simulated bodies. The coupling, however, allows for a very efficient collision detection system, tailored to the needs of dynamic simulation.

A block diagram of *Impulse's* entire collision detection system is shown in Figure 2.1. The Lin-Canny algorithm is used to compute geometric information about the

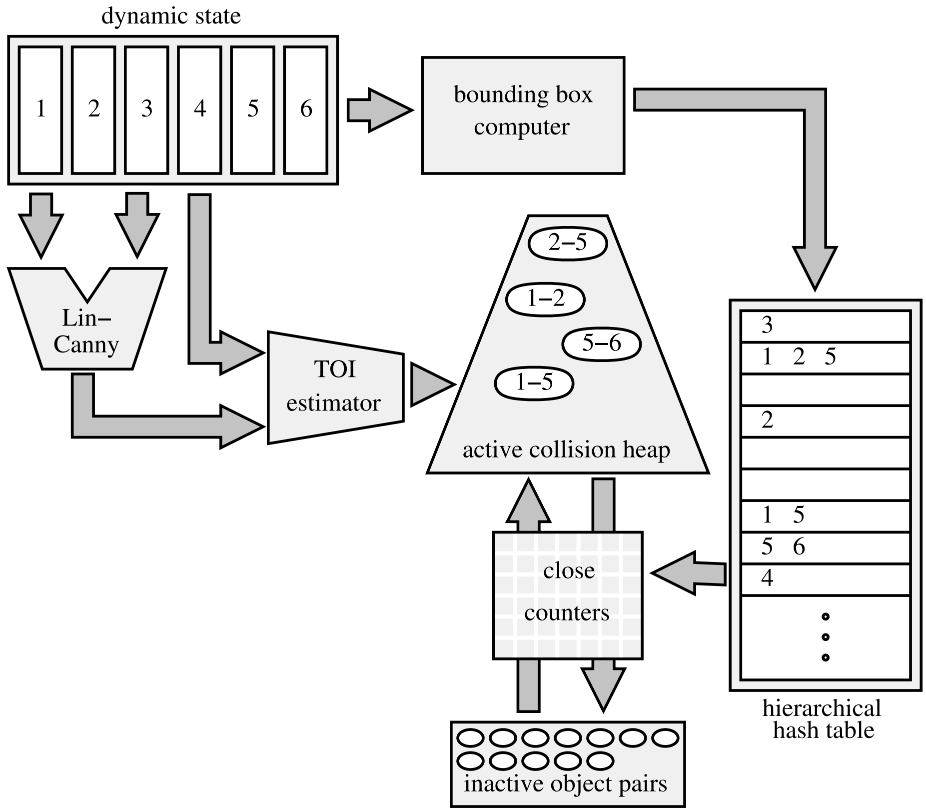

Figure 2.1: *A block diagram of* Impulse's *collision detection system. Arrows represent data flows. Numbers designate indices of bodies in the simulation. The Lin-Canny algorithm, TOI estimator, and active collision heap perform narrow phase collision detection; the bounding box computer, hierarchical hash table, and close counters are used for the broad phase.*

separation of a pair bodies; it is described in detail in Section 2.2. This geometric information, plus the dynamic state of the bodies is used to estimate a time of impact (TOI) for the pair. Pending collision checks between pairs of bodies are maintained in the collision heap, sorted by the pair's TOI. Time of impact estimation and the collision heap are discussed in Section 2.3. The bounding box computer, hierarchical hash table, and close counters form the broad phase collision detection system. They typically prune most of the pairs of bodies from the active collision heap using inexpensive tests. These components are described in Section 2.4. The section also presents a comparison between the spatial hashing technique used in *Impulse's* broad phase, and a common technique based on coordinate sorting. Throughout most of the chapter, the focus is on collision detection between ballistic bodies, which are ubiquitous in impulse-based simulation. Section 2.5 describes how the algorithms may be extended to more general types of motion.

2.2 The Lin-Canny algorithm

The heart of *Impulse's* collision detection scheme is the Lin-Canny closest features algorithm [Lin93, LC91], an extremely fast method for tracking the features (faces, edges, or vertices) between a pair of convex polyhedra moving through space. The principle behind the algorithm is best described with a two-dimensional example. A fundamental concept in the Lin-Canny algorithm is that of a Voronoi region. Consider the polygon shown in Figure 2.2. The polygon has eight features: four vertices and four edges. For each

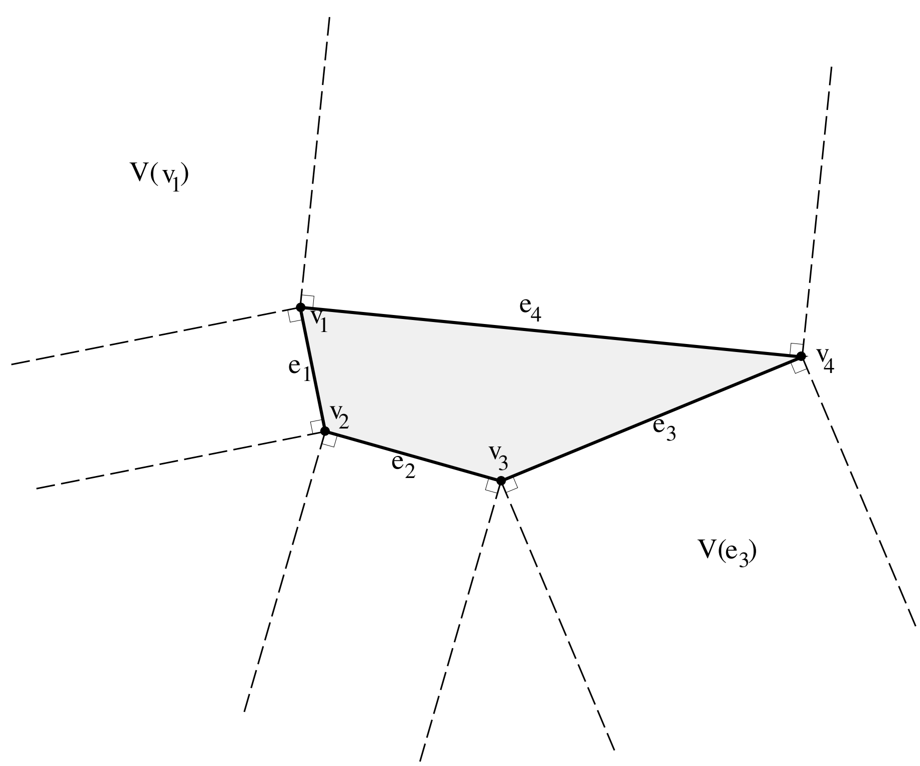

Figure 2.2: *A polygon and its Voronoi regions.*

feature F, the set of points closer to F than to any other feature of the polygon is called the *Voronoi region* of F, and denoted $V(F)$. The shapes of the Voronoi regions are easily deduced for polygonal bodies. From each vertex, extend two rays outward from the polygon, each perpendicular to one of the edges incident to the vertex. These rays form boundaries between the Voronoi regions. The Voronoi region of a vertex is the infinite cone lying between the two rays emanating from that vertex. The Voronoi region of an edge is the semi-infinite rectangle lying between two parallel rays passing through the edge's endpoints. Collectively, the Voronoi regions partition the space outside the polygon.

Theorem 1 *Given non-intersecting polygons A and B, let* **a** *and* **b** *be the closest points between feature F_a of A, and feature F_b of B, respectively. If* **a** *and* **b** *are the closest points between A and B, then $\mathbf{a} \in V(F_b)$ and $\mathbf{b} \in V(F_a)$.*[1]

Proof: Suppose $\mathbf{a} \notin V(F_b)$. Then **a** is in some other Voronoi region, say $V(F_c)$, and **a** is closer to F_c than to any other feature on B. Since $\mathbf{b} \in F_b$, $\mathbf{b} \notin F_c$, and so **a** and **b** can not be closest points. A similar results holds if $\mathbf{b} \notin V(F_a)$. □

The fundamental basis of the Lin-Canny algorithm is the converse of Theorem 2, which is true for convex objects (see [Lin93] for the proof).

Theorem 2 *Given non-intersecting* convex *polygons A and B, let* **a** *and* **b** *be the closest points between feature F_a of A, and feature F_b of B, respectively. If $\mathbf{a} \in V(F_b)$ and $\mathbf{b} \in V(F_a)$, then* **a** *and* **b** *are the closest points between A and B.*

Theorem 2 suggests an algorithm for finding the closest points between convex polygons. Consider the situation on the left of Figure 2.3. Here, closest point candidates **a** and **b**

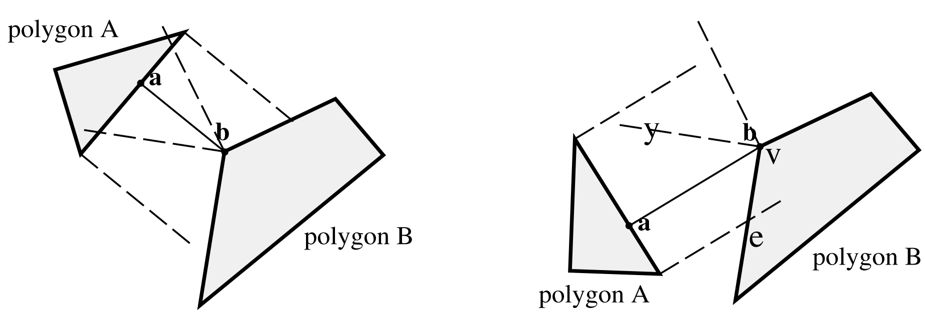

Figure 2.3: Left: *Theorem 2 implies* **a** *and* **b** *are the closest points between A and B.* Right: $\mathbf{a} \notin V(F_b)$, *and so* **a** *and* **b** *are no longer closest points. The closest feature on B will be updated from vertex v to edge e.*

[1]For simplicity, degenerate cases where the points are on the boundary of Voronoi regions are ignored here. [Lin93] for more details.

each lie in the Voronoi region of the other's containing feature. By Theorem 2, they are the closest points. In the situation depicted on the right, **b** is still in the Voronoi region of F_a, however, **a** is no longer in the Voronoi region of F_b. Specifically, **a** lies on the wrong side of ray y. In this case, the Lin-Canny algorithm specifies that feature F_b should be updated to the feature on the other side of ray y, namely the edge e. At this point, the closest points between the new features are computed, and the Voronoi check is made again. This process can continue for many iterations, however it is guaranteed to eventually terminate with $\mathbf{a} \in F_b$, $\mathbf{b} \in F_a$, and **a** and **b** the closest points between A and B. The three-dimensional version of the algorithm is a natural extension of the two-dimensional case. The bodies in question are polyhedra, the features are vertices, edges, and faces, the Voronoi regions are infinite regions of space bounded by constraint planes rather than rays (Figure 2.4). The

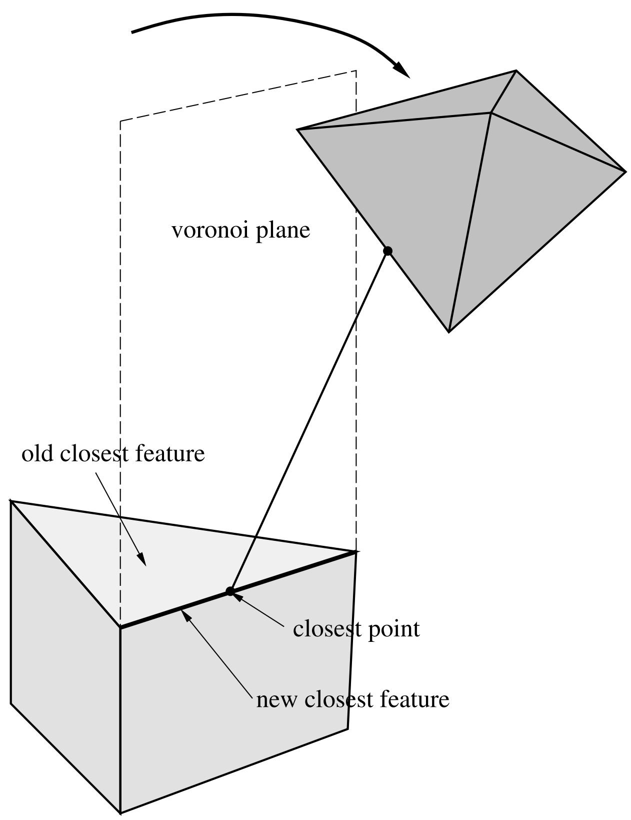

Figure 2.4: *Tracking closest features of polyhedra with the Lin-Canny algorithm.*

basic algorithm remains the same. For details, see [Lin93, LC91].

Although designed to track the closest features, the Lin-Canny algorithm is easily extended to a collision detection algorithm. The distance between two polyhedra is computable from simple geometric formulas, given the closest features. The closest points are obtained as a by-product of these calculations. With finite precision arithmetic, a *collision epsilon* ε_c must be used. The collision detection system reports a possible collision when the inter-polyhedral distance falls below ε_c. The particular value of ε_c is not critical; a value is chosen based on how large a gap is tolerable. For animation purposes, the collision epsilon should be smaller than a pixel width, so that the objects appear to touch when colliding. For the simulation examples described later in the thesis, ε_c was two to three orders of magnitude smaller than the dimensions of the objects. For example, in the bowling simulation, which used a standard 60 foot alley and 15 inch pins, ε_c was one millimeter.

2.2.1 Collision detection and coherence

For efficient collision detection for simulation, it is extremely important to take advantage of *geometric coherence* (also called *temporal* or *frame-to-frame coherence*). The problem instances presented to the collision detection algorithm are a series of closely related problems. In Lin-Canny terms, the closest features between a given pair of objects usually change relatively infrequently. Even if the features are changing upon every invocation of the algorithm, due to highly discretized polyhedral models or high velocities, the pair of closest features from the last invocation of the algorithm is a good starting point for the search for the current pair of closest features. In *Impulse*, a two-dimensional table of closest feature pairs is maintained. For every pair of bodies in the simulation, there is a corresponding table entry containing the closest feature pair for these bodies, computed from the last invocation of the algorithm. Figure 2.5 illustrates the effect of coherence on tracking the closest features.

The Lin-Canny algorithm has been described as taking "expected constant" time to report a pair of closest features. This claim stems from coherence; often the closest features do not change between successive calls, and the algorithm verifies this fact in constant time. This is a bit misleading. Consider tracking closest features between a small satellite orbiting the Earth, over its equator. If the Earth is modeled as a tessellated sphere with N facets, then during one orbit of the satellite, tracking on the Earth must progress through $O(\sqrt{N})$ features. As the resolution of the Earth model increases, more work is clearly

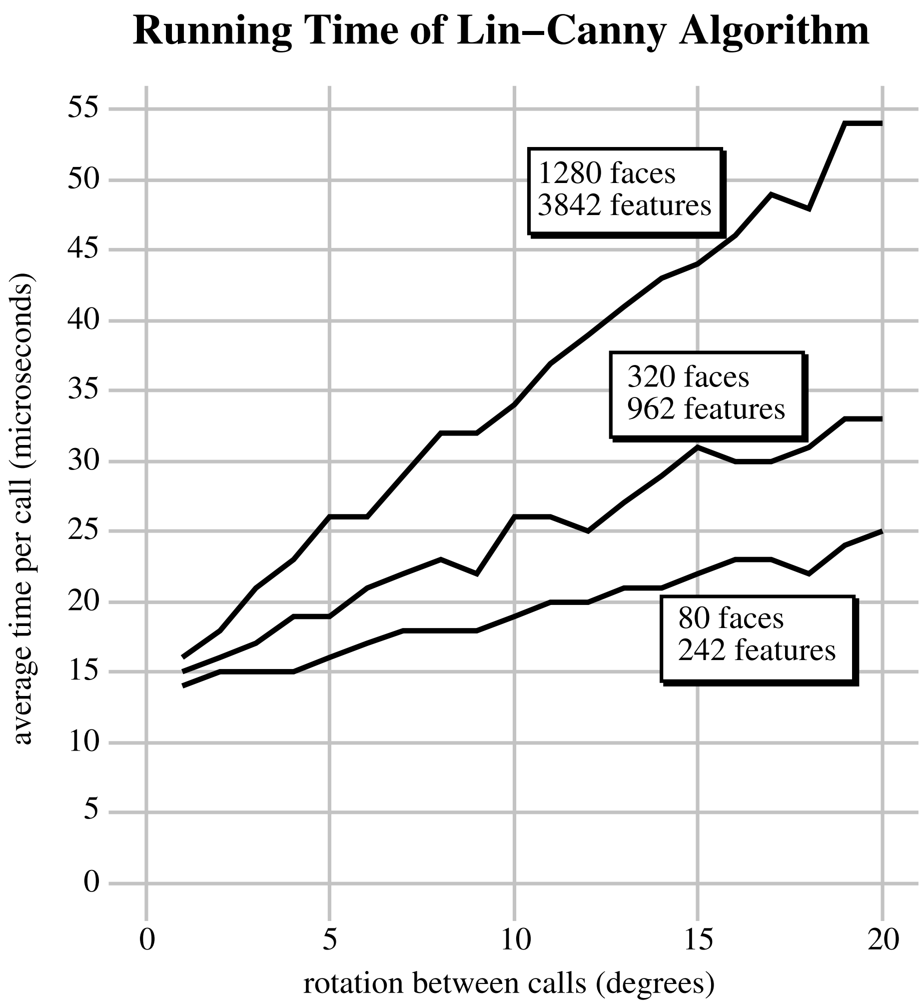

Figure 2.5: *This graph shows the effect of coherence on the performance of the Lin-Canny algorithm. The algorithm was used to track the closest features between a fixed cube and a polyhedral model of a sphere as the sphere rotated on an axis parallel to the nearest surface of the cube. The amount of sphere rotation between successive calls to the algorithm was varied from one to 20 degrees, in one degree steps. This experiment was performed for three different discretization resolutions for the sphere, as indicated above. The performance of the algorithm decreases as the rotation speed increases, due to a decrease in coherence. Also note the insensitivity of the algorithm to polyhedron complexity, when coherence can be exploited. At a rotational speed of one degree between calls, a 16-fold increase in complexity results in a 15% increase in execution time.*

being done to track the closest feature as it circumnavigates the planet, even if the satellite speed remains constant. In this case, the Lin-Canny algorithm is $O(\sqrt{N})$. Figure 2.5 and Graph 2 in Cohen, *et. al.* [CLMP95] also illustrate that the running time of the Lin-Canny algorithm depends on the number of features. One difficulty of assigning a complexity to the algorithm is that it is very dependent on how the objects are moving. If the satellite mentioned above falls straight down toward Earth, the algorithm is again constant time. The claim of "expected constant" time raises more questions than it answers, however, and *almost constant* time is a better description. In the satellite example, the coefficient on $\sqrt{N}$ is probably extremely small compared to the constant term. In experiments with *Impulse*, there is negligible slowdown in simulation speed, when polyhedral models of spheres with a few hundred facets are replaced with polyhedral models with over 20,000 facets (over 60,000 features). The latter models are used in many of the simulations described later in the thesis.

2.2.2 Extensions to the Lin-Canny algorithm

The extension of the basic Lin-Canny algorithm to curved objects has been studied by Lin and Manocha [LM93]. Curved bodies are approximated with a polyhedral mesh, and closest points are tracked between these meshes. The closest points on the meshes are projected onto the actual curved surfaces, and a numerical root finding method uses these points as a starting point to locate the true closest points. A general form of this algorithm has not been implemented.

Another extension, used in *Impulse*, is the extension to non-convex bodies. Assuming non-convex bodies can be decomposed into a group of convex pieces, Lin-Canny can still be used to compute the distance between a pair of bodies. If body A is decomposed into m convex pieces, $A_1, \ldots, A_m$, and body B is decomposed into n convex pieces, $B_1, \ldots, B_n$, then the distance between A and B can be computed as

$$d(A, B) = \min_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} d(A_i, B_j) \tag{2.1}$$

In other words, the distance computation is broken into $m \times n$ standard Lin-Canny computations between convex bodies. Because of this reduction of the non-convex case to a group of convex cases, the rest of this chapter assumes convex bodies.

Significant improvement over this naive scheme is possible by computing the convex hulls of each of the non-convex bodies, and computing the distance between bodies as the distances between their hulls. This requires only one Lin-Canny invocation, and since A is enclosed by its convex hull, the distance to the convex hull is a lower bound (a conservative estimate) on the true distance to A. Only if the distance between the hulls reaches zero, must the bodies be unwrapped and treated as a collections of convex pieces. For complex bodies, this scheme can be applied recursively to obtain an entire tree structure for a non-convex body A. The inner nodes of the tree are convex hulls of various subsets of the entire body, and if a particular convex hull is pierced, that tree node is replaced by its children. The leaves of the tree represent the underlying convex decomposition of the entire body, and the root of the tree is the convex hull of the entire body. With such a structure, only as much of the body is unwrapped as needs to be to determine distance; the rest of the concavities remained wrapped in convex hulls. Details of this scheme may be found in Ponamgi, *et. al.* [PML95]. At present, a general version of this algorithm does not exist, although one is being developed as part of *I-COLLIDE* [CLMP95]. In *Impulse*, the naive approach represented by Equation (2.1) is used to handle non-convex bodies.

2.3 Prioritizing collision checks

For simulation, the collision detection algorithm must determine the time t_c at which a collision occurs between bodies. The typical approach is akin to the approach taken in numerical root finding. Assume there exists a function that takes two bodies and returns a boolean value indicating if they are penetrating or not. If the bodies are not penetrating at time t_0, dynamic integration of the system state continues to some time t_1. If at this point, the bodies are re-tested and penetration is detected, than a collision has occurred between them at some time t_c with $t_0 \leq t_c \leq t_1$. In this case, a new point $t_m \in (t_0, t_1)$ is chosen, and dynamic integration is performed from t_0 to t_m. Based on the result of a collision test at the time t_m, the process is repeated recursively on the interval (t_0, t_m), or (t_m, t_1). The process is repeated until the width of the interval falls below some tolerance. The time t_m may be chosen as the midpoint of the interval (t_0, t_1), however Baraff reports much faster convergence using an interpolation method instead of simple interval bisection [Bar89]. If the penetration function returns a distance of separation or penetration, rather than a simple boolean value, one can more accurately estimate the moment of collision by examining this distance at t_0 and t_1. The distance of separation is well defined for convex bodies; distance of penetration can be defined as in [SSCK94].

A particular trait of the Lin-Canny closest features algorithm (as originally specified) had a strong impact on the way collision detection is performed in *Impulse*. When the algorithm is passed two polyhedra that are penetrating, it cycles between pairs of closest features, never terminating. The proofs which guarantee convergence and termination rest on the assumption that the bodies are non-penetrating. Lin motivates the solution to this problem, based on constructing internal *pseudo-Voronoi* regions that partition the space inside of the polyhedron, and matching each interior region with a feature on the polyhedron [Lin93]. This extension to the original algorithm is nontrivial, and it was not pursued at the time the collision detection subsystem of *Impulse* was developed.

Instead, the collision detection subsystem in *Impulse* was designed to detect collision *before* it occurred. This is done by obtaining successively closer approximations to the exact collision time t_c, without ever integrating beyond t_c. The situation is as depicted in Figure 2.6, in which the function $d(t)$ represents the distance between two bodies; negative values and zero crossings correspond to penetration and collisions, respectively. The

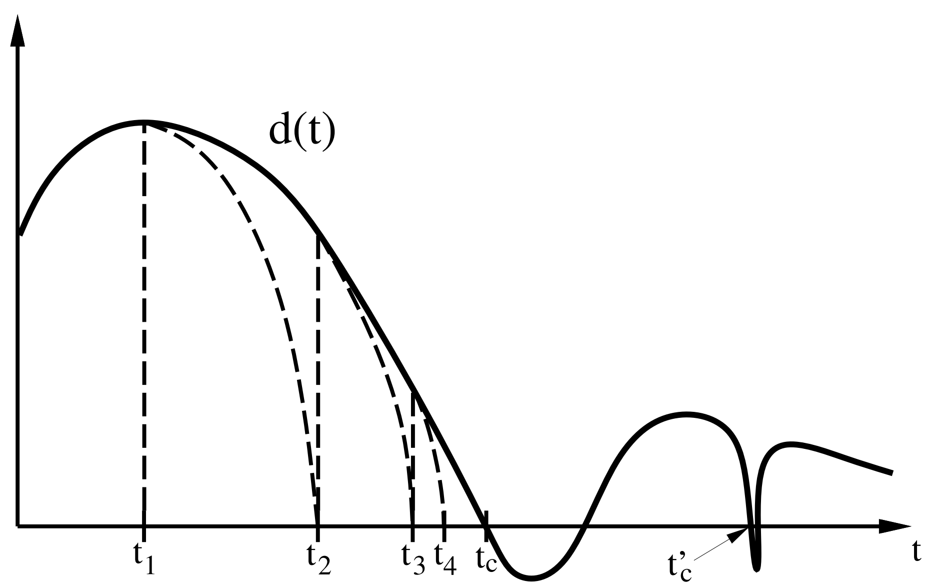

Figure 2.6: *A one-sided approach to root finding is used to find collision times in* Impulse. *The function $d(t)$ is the separation distance between two bodies over time; the zero at t_c indicates a collision. The velocities of the two bodies at time t_i are used to compute a parabola that is less than $d(t)$ for $t \geq t_i$. The next zero crossing of the parabola gives t_{i+1}, a closer approximation to t_c that is guaranteed not to exceed t_c.*

derivative of $d(t)$ is a measure of the velocity of approach of the bodies. In *Impulse*, a series of approximations $t_1, t_2, \ldots, t_n$ is made to the actual collision time t_c, such that no t_i exceeds

t_c. At each t_i, the current velocities of the relevant bodies are used to compute a second order function which is a lower bound on the separation distance between the bodies, for $t \geq t_i$. The abscissa of the next zero crossing of the corresponding parabola becomes t_{i+1}. The series of closer approximations to t_c ends when the distance between the bodies falls below ε_c. Using a Newton-Rhapson approach to root finding, it is possible to miss a zero crossing in $d(t)$; for example, the crossing at t'_c in Figure 2.6 is likely to be missed. This root can also be missed if there is a minimum time interval between collision checks. With the one-sided approach, every root of $\delta(t)$ is found.

2.3.1 The collision heap

When collision checking reveals that the distance between two bodies exceeds the collision epsilon ε_c, a lower bound on their time of impact (TOI) is automatically computed and returned. In terms of Figure 2.6, the current time is t_i, and the value returned is t_{i+1}. Assume a TOI estimator exists, which takes as inputs the dynamic states of two bodies, along with the closest points between them. It returns a lower bound on the time of impact, based on the assumption that the bodies will follow ballistic trajectories until the moment of impact. The returned TOI estimates can be used to adaptively control the frequency of collision checks while integrating the dynamic system.

Collision checks are scheduled in a *collision heap.* Corresponding to each pair of bodies in the simulation is an element in the heap, containing a field with the last computed TOI for that pair. The TOI field is a lower bound on the true time of impact for the bodies, which is unknown. The heap is sorted on this TOI field, so that no collisions can occur before the time in the TOI field of the top heap element (Figure 2.7). The top level simulation loop is very simple. The system is evolved to the time in the TOI field of the top heap element, at which point collision checking is performed for this top top pair of bodies. If the distance between the bodies is below the collision epsilon, a collision is declared, and is handled by the collision resolution system. In any case, the TOI for the body pair is recomputed, possibly causing it to drop down in the heap, and the process is repeated. This scheme adapts the frequency of collision checks appropriately: when a pair of bodies are far apart or moving slowly, collision checks between them will be infrequent; as the bodies approach, checks increase as necessary.

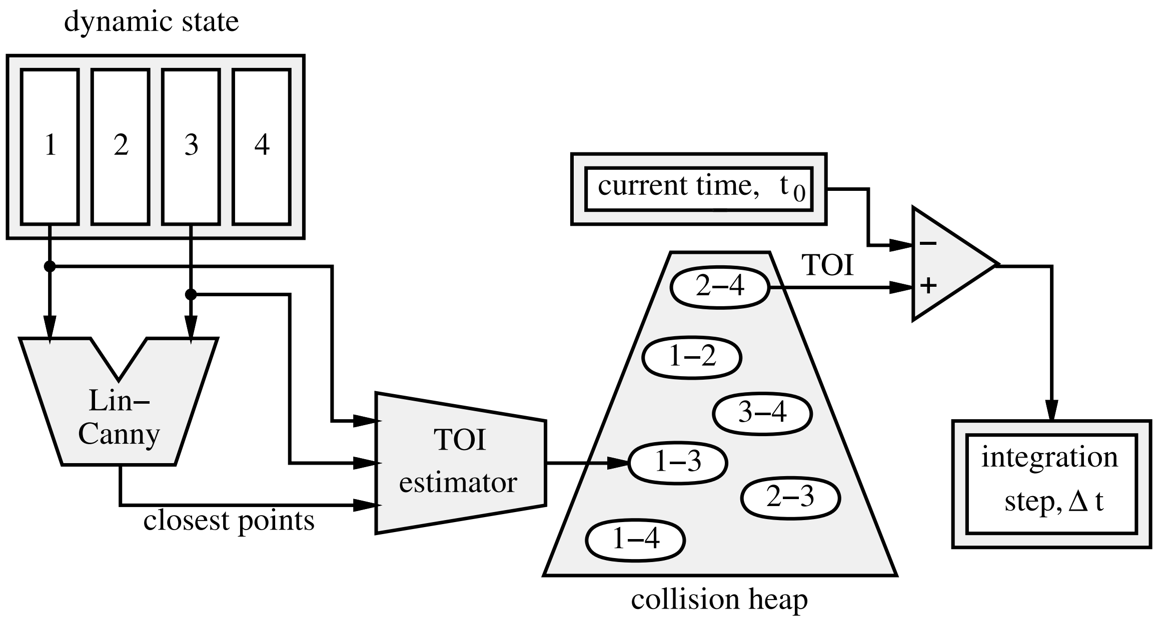

Figure 2.7: Impulse's *narrow phase collision detection system prioritizes collision checks in a heap, based on estimated times of impact. The time of impact of the top heap element determines the size of the next integration step. Numbers designate the indices of the different bodies in the simulation.*

2.3.2 Estimating time of impact

For the remainder of this chapter, the following notation is used:

- t — A general time variable. The positions and velocities of all bodies are functions of t.
- t_0 — The current time in the simulation. The positions and velocities of all bodies are known at time t_0.
- t_c — The time of the next collision between two bodies.
- Δt — Some time step into the future. Often the motion of bodies over the interval $[t_0, t_0 + \Delta t]$ is reasoned about.

The key problem that must be solved in order to apply the collision heap scheme is:

Problem 1 Given: *The current positions and velocities of two ballistic, convex bodies, and the closest points between the bodies.* Compute: *A lower bound on the time of impact of the two bodies, assuming they continue their current ballistic trajectories.*

One difficulty is that the points on the bodies that will ultimately collide are not readily found from the given information.[2] Figure 2.8 illustrates a bad case: the linear and angular velocities of the bodies are such that the closest points (with total velocities $\mathbf{u}_1$ and $\mathbf{u}_2$) are moving apart. However, another pair of points, not even considered by the closest features algorithm, are almost as close as the closest points, and are approaching each other quickly.

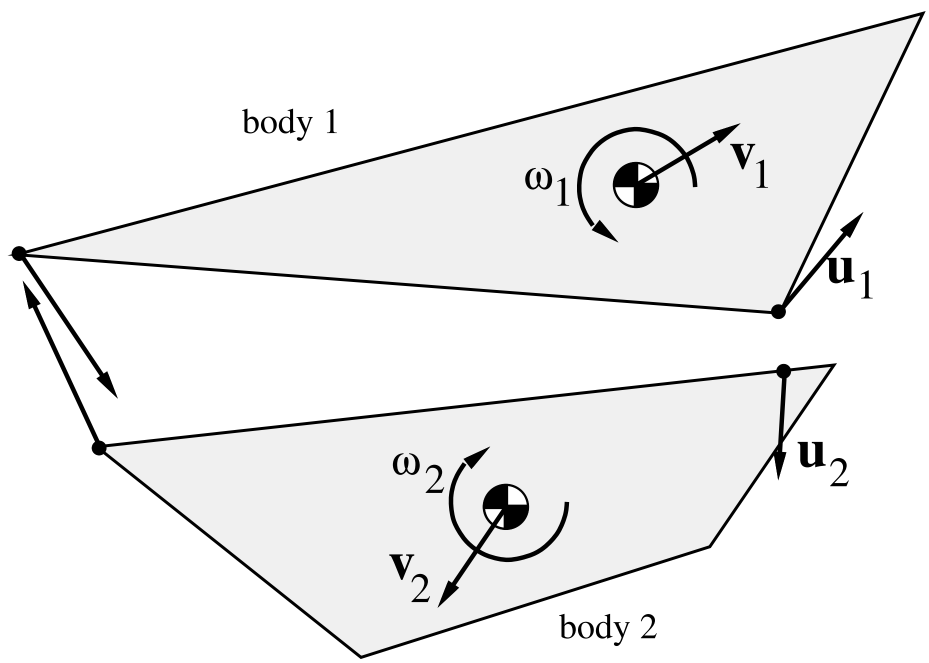

Figure 2.8: *The velocities of the closest points only is insufficient for predicting time of impact.*

One method of bounding the time of impact relies on the convexity of the bodies. Consider the bodies shown in Figure 2.9. Call the closest points on the two bodies $\mathbf{c}_1$ and $\mathbf{c}_2$, and let $\mathbf{d} = \mathbf{c}_2 - \mathbf{c}_1$. Since body 1 is convex, it must lie entirely on one side of the plane passing through $\mathbf{c}_1$ and perpendicular to the vector $\mathbf{d}$. The same is true for body 2 and the corresponding plane through $\mathbf{c}_2$. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be the points at which the bodies will ultimately collide. No matter where these points are located and what their path to

[2]There are closed form solutions for the orientation of a rigid body as a function of time that use elliptic integrals [MR94]. Using these equations with some method of evaluating elliptic integrals might lead to tighter bounds than the ones discussed here.

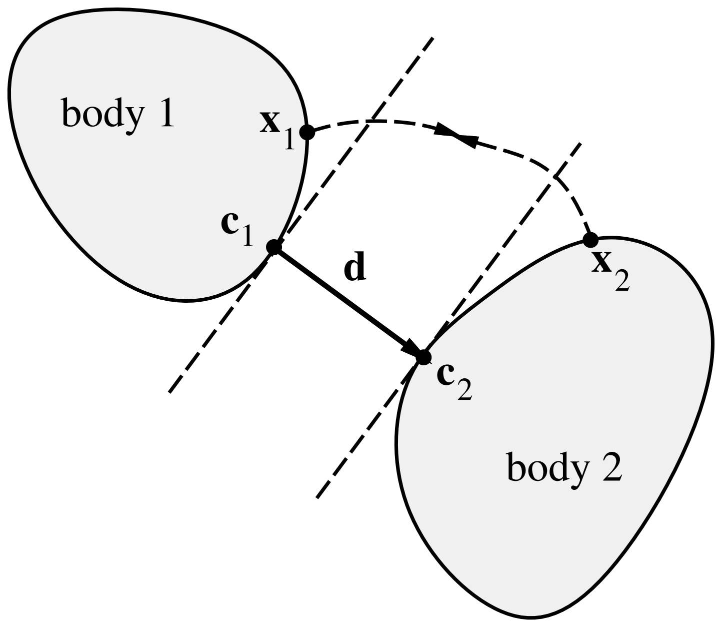

Figure 2.9: *Finding a lower bound on the time of impact of convex bodies.*

impact will be, they will have to cover the distance $d = \|\mathbf{d}\|$ in the direction of $\mathbf{d}$. Let $\hat{\mathbf{d}}$ be a unit vector along $\mathbf{d}$. Let $D_1(t)$ be an upper bound on the distance traveled by *any* point on body 1 along $\hat{\mathbf{d}}$ during the interval $[t_0, t]$. Let $D_2(t)$ similarly bound the distance traveled by *any* point on body 2 along $-\hat{\mathbf{d}}$. If a collision occurs at time t_c,

$$D_1(t_c) + D_2(t_c) \geq d.$$

A lower bound on the time of impact can be found by replacing this inequality with an equality, and solving for t_c. To facilitate this, the functions $D_i(t)$ should be kept simple; *Impulse* uses quadratic functions.

Problem 2 (TOI coefficients) Given: *The position and velocity of a body at the current time t_0, and a direction specified by the unit vector $\hat{\mathbf{d}}$.* Find: *Coefficients A and B such that the distance traveled by* any *point on the body in the direction $\hat{\mathbf{d}}$ over the interval $[t_0, t]$ satisfies*

$$D(t) \leq A(t - t_0)^2 + B(t - t_0).$$

Consider the case of a ballistic body. The total velocity $\mathbf{u}(t)$ of an arbitrary point $\mathbf{x}$ on the body is given by

$$\mathbf{u}(t) = \mathbf{v}(t) + \boldsymbol{\omega}(t) \times \mathbf{r}(t).$$

Here, $\mathbf{v}$ and $\boldsymbol{\omega}$ are the linear center of mass velocity and angular velocity of the body, and $\mathbf{r}$ is the position vector from the body's center of mass to $\mathbf{x}$, all specified in a fixed reference frame. Since the body is ballistic,

$$\mathbf{v}(t) = \mathbf{v}(t_0) + \mathbf{g}(t - t_0),$$

where $\mathbf{g}$ is the vector acceleration of gravity in the reference frame. Thus, the velocity of $\mathbf{x}$ in the direction $\hat{\mathbf{d}}$ is given by

$$\mathbf{u}(t) \cdot \hat{\mathbf{d}} = [\mathbf{v}(t_0) + \mathbf{g}(t - t_0)] \cdot \hat{\mathbf{d}} + \boldsymbol{\omega}(t) \times \mathbf{r}(t) \cdot \hat{\mathbf{d}}.$$

Letting $r_{\max}$ be the maximum distance of *any* point on the body from the center of mass, and $\omega_{\max}$ be the maximum magnitude of the body's angular velocity during the current ballistic phase,

$$\mathbf{u}(t) \cdot \hat{\mathbf{d}} \leq [\mathbf{v}(t_0) + \mathbf{g}(t - t_0)] \cdot \hat{\mathbf{d}} + r_{\max}\omega_{\max}.$$

The quantity $r_{\max}$ may be pre-computed and stored for each body. The next section discusses the computation of $\omega_{\max}$. Integrating the above equation over time,

$$\int_{t_0}^{t} \mathbf{u}(\tau) \cdot \hat{\mathbf{d}} \, d\tau \leq \frac{1}{2}(\mathbf{g} \cdot \hat{\mathbf{d}})(t - t_0)^2 + \left[\mathbf{v}(t_0) \cdot \hat{\mathbf{d}} + r_{\max}\omega_{\max}\right](t - t_0).$$

The function on the right hand side is a suitable choice for $D(t)$. For ballistic bodies, the TOI coefficient routine returns

$$A = \frac{1}{2}\mathbf{g} \cdot \hat{\mathbf{d}} \tag{2.2}$$

$$B = \mathbf{v}(t_0) \cdot \hat{\mathbf{d}} + r_{\max}\omega_{\max}. \tag{2.3}$$

Other types of motion besides ballistic motion can be accommodated by the same scheme, and are discussed later.

To compute a lower bound on the time of impact between body 1 and body 2, a TOI coefficient routine is called for each of the bodies (In computing the TOI coefficients for body 2, $-\hat{\mathbf{d}}$ is used in place of $\hat{\mathbf{d}}$). Call the coefficients for body 1 A_1 and B_1; call those for body 2 A_2 and B_2. The lower bound TOI is the smallest real root $t > t_0$ of the quadratic equation

$$(A_1 + A_2)(t - t_0)^2 + (B_1 + B_2)(t - t_0) = d.$$

If both bodies are ballistic, a common case in impulse-based simulation, the gravitational contributions for each body cancel, and $A_1 = -A_2$. In this case, only a linear equation must be solved. If the above equation has no real roots greater than t_0, then the bodies can never collide, given their current trajectories. In this case, the time of impact routine returns infinity. If one of the bodies collides with some other body in the environment, however, the time of impact between bodies 1 and 2 must be recomputed and may be reduced to a finite value.

2.3.3 Bounding ballistic angular velocity

The TOI calculations of the last section assume knowledge of the maximum magnitude of the angular velocity of a body during a ballistic trajectory.

Problem 3 Given: *The current angular velocity* $\boldsymbol{\omega}$ *of a ballistically moving body.* Find: *A bound on the maximum magnitude of* $\boldsymbol{\omega}$ *while the body moves in its current trajectory.*

Since only the magnitude of the body's angular velocity vector is needed, $\boldsymbol{\omega}$ can be expressed in the most convenient frame, in this case, the body frame. Below, vectors and tensors are expressed in this frame, unless otherwise noted.

The angular momentum of the body is given by

$$\mathbf{L}(t) = \mathbf{I}\boldsymbol{\omega}(t),$$

where $\mathbf{I}$ is the body frame inertia tensor (see Appendix A.3). There is no time dependence in this matrix. Since the only external force acting on the body is gravity, which acts through the center of mass, the angular momentum is conserved in the inertial frame. Although angular momentum is not conserved in the body frame, it is related to the constant inertial angular momentum through a time-varying rotation matrix. Hence, the magnitude of the body angular momentum is conserved. Therefore,

$$\|\mathbf{L}(t_0)\| = \|\mathbf{I}\boldsymbol{\omega}(t)\|.$$

The body inertia tensor $\mathbf{I}$ is diagonal. Calling its diagonal elements I_x, I_y, and I_z, the above equation can be rewritten

$$\frac{\omega_x(t)^2}{\left(\frac{\|\mathbf{L}(t_0)\|}{I_x}\right)^2} + \frac{\omega_y(t)^2}{\left(\frac{\|\mathbf{L}(t_0)\|}{I_y}\right)^2} + \frac{\omega_z(t)^2}{\left(\frac{\|\mathbf{L}(t_0)\|}{I_z}\right)^2} = 1,$$

Thus, the body angular velocity vector is constrained to lie on an ellipsoid in $\mathbb{R}^3$ (Figure 2.10). The maximum value of $\|\boldsymbol{\omega}(t)\|$ is just the length of the semi-major axis of this

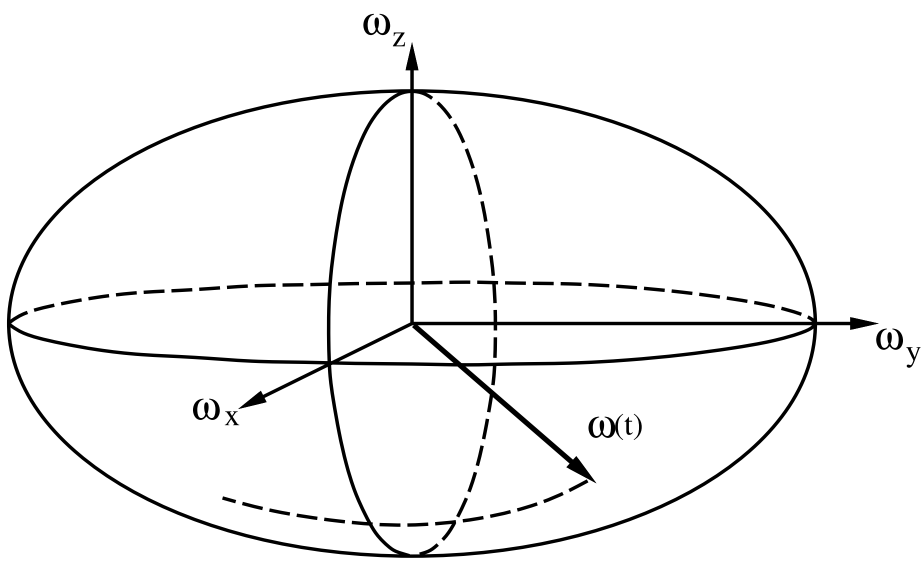

Figure 2.10: *The body angular velocity vector remains on an ellipsoid over a ballistic trajectory.*

ellipsoid, thus

$$\|\boldsymbol{\omega}(t)\| \leq \frac{\|\mathbf{L}(t_0)\|}{\min(I_x, I_y, I_z)} = \frac{\sqrt{I_x^2\omega_x(t_0)^2 + I_y^2\omega_y(t_0)^2 + I_z^2\omega_z(t_0)^2}}{\min(I_x, I_y, I_z)}. \tag{2.4}$$

This bound was reported in [MC95b], however a slight improvement is possible. The conservation of energy defines a different ellipsoid on which the angular velocity must lie. Since there is no net torque acting on a body in a ballistic trajectory, the rotational kinetic energy is constant, given by

$$E = \frac{1}{2}\boldsymbol{\omega}(t)^T \mathbf{I}\boldsymbol{\omega}(t).$$

Since the body inertia tensor is diagonal,

$$2E = I_x\omega_x(t)^2 + I_y\omega_y(t)^2 + I_z\omega_z(t)^2.$$

Writing this in the general form of an ellipsoid gives

$$\frac{\omega_x(t)^2}{\sqrt{\frac{2E}{I_x}}^2} + \frac{\omega_y(t)^2}{\sqrt{\frac{2E}{I_y}}^2} + \frac{\omega_z(t)^2}{\sqrt{\frac{2E}{I_z}}^2} = 1.$$

The maximum value of $\|\boldsymbol{\omega}(t)\|$ is the length of the semi-major axis of this ellipsoid:

$$\|\boldsymbol{\omega}(t)\| \leq \sqrt{\frac{2E}{\min(I_x, I_y, I_z)}} = \sqrt{\frac{I_x\omega_x(t_0)^2 + I_y\omega_y(t_0)^2 + I_z\omega_z(t_0)^2}{\min(I_x, I_y, I_z)}}. \tag{2.5}$$

Let $I_{\min} = \min(I_x, I_y, I_z)$ and $I_{\max} = \max(I_x, I_y, I_z)$. The ratio of the angular kinetic energy bound (2.5) to the angular momentum bound (2.4) is

$$r = \sqrt{\frac{[I_x\omega_x(t_0)^2 + I_y\omega_y(t_0)^2 + I_z\omega_z(t_0)^2]I_{\min}}{I_x^2\omega_x(t_0)^2 + I_y^2\omega_y(t_0)^2 + I_z^2\omega_z(t_0)^2}}.$$

This implies

$$\sqrt{\frac{I_{\min}}{I_{\max}}} \leq r \leq 1. \tag{2.6}$$

The angular kinetic energy bound is always at least as tight as the angular momentum bound. The bounds are the same when the body is rotating exactly about the axis of minimum inertia, and the bounds are most different when the rotation is exactly about the axis of maximum inertia. The two bounds are always equivalent if the diagonal entries of the body inertia tensor are all equal, as is the case for a uniform density sphere or cube.

2.4 Bounding box techniques

The TOI estimation described above assumes that bodies follow ballistic trajectories between collisions. Suppose collision detection is performed on bodies 1 and 2, indicating they are not yet colliding, and let t_c be the computed time of impact for these two bodies. The time t_c reflects the soonest time these bodies may collide, assuming they continue along their current ballistic trajectories. It is possible, however, that body 1 may collide with some other body, say body 3, before t_c is reached. A collision impulse will be applied to body 1, sending it on a new ballistic trajectory. Thus, t_c is invalid, and must be recomputed. In fact, all TOIs that involve body 1 or body 3 must be recomputed. The collision heap is actually a priority queue, since the keys are not static, but can increase or decrease, causing heap elements to rise or fall before ever reaching the top.

Early versions of *Impulse* performed $O(n)$ TOI updates upon every collision in an n body simulation, but the method scales poorly. Consider a group of coins tossed onto a large, flat surface. Even if the coins are separated by large distances, as they begin to settle on the surface, collisions become frequent. Every time a coin collides with the surface, the TOIs between that coin and every other coin must be recomputed, even when the coins are in no danger of colliding. The problem is that spatial locality is not being exploited. When a coin experiences a collision with the surface, it is reasonable to recompute its TOI with nearby coins, but not with ones that are far away. A related inefficiency is that collision checks are performed between bodies very far apart and in no danger of colliding. The heap scheme reduces the frequency of checks between such bodies, however they are still made at regular intervals. Bounding box techniques may be used to reduce these superfluous collision checks and TOI updates. The method used in *Impulse* is based on a hashing scheme proposed by Overmars for solving point location problems [Ove92]. The technique is first discussed in a static context, and then the extension to moving bodies is described.

2.4.1 Finding static box intersections

Point location problems occur frequently in computational geometry. One variant is expressed as follows:

Problem 4 (Point Location) *Given a number of non-intersecting cells in space, store the arrangement such that for a given query point p, the cell containing p (if any) can be determined efficiently.*

Here, a *cell* is a connected region of space. Overmars presents two solutions to this problem under the restriction that the cells are *fat* [Ove92]. The more efficient solution involves surrounding each cell by an axes-aligned bounding box, and storing the location of these boxes in a hash table. The technique can be extended to solve a more useful problem for collision detection:

Problem 5 (Static box intersections) *Given n axes-aligned rectangular boxes $B_1, \ldots, B_n$, fixed in space, store this arrangement such that the boxes that intersect a specified query box, B_q can be determined efficiently.*

To attack this problem, consider partitioning space into a cubical tiling with resolution ρ. Any point (x, y, z) in space belongs to a unique tile, specified by integer coordinates, under the tiling map τ:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \xrightarrow{\tau} \begin{bmatrix} \lfloor x/\rho \rfloor \\ \lfloor y/\rho \rfloor \\ \lfloor z/\rho \rfloor \end{bmatrix} \tag{2.7}$$

The tiles that box B_i intersects are found by computing the images under τ of two of B_i's corners: the one of minimum x, y, and z coordinates, and the one of maximum x, y, and z coordinates. The two tiles containing these corners, and the other tiles "between" them, are the tiles that intersect B_i. A tile with coordinates (a, b, c) is between the two tiles with coordinates (a^-, b^-, c^-) and (a^+, b^+, c^+) if and only if $a^- \leq a \leq a^+$, $b^- \leq b \leq b^+$, and $c^- \leq c \leq c^+$. There are an infinite number of tiles in unbounded space, but only a finite number that are intersected by at least one box. For each tile that a box intersects, the box's label is stored in the hash table, hashed under the tile's integer coordinates. The query box B_q can also be mapped to a set S of hash buckets according to which tiles it overlaps. The union of all boxes whose labels appear in buckets of S can be quickly determined, and this set forms a candidate set of boxes for Problem 5. The candidate set is further checked against B_q using more expensive box intersection tests.

How to choose the tiling resolution ρ is not obvious, and in fact this can be problematic when the sizes of the boxes vary widely. If ρ is small, the larger boxes may intersect a huge number of tiles, and thus require a large amount of storage in the hash table. In addition, when the static assumption is relaxed, updating the positions of these large, moving boxes will be inefficient. On the other hand, if ρ is large, the tiling will have poor resolution power for the smaller boxes. Many small boxes may hash to the same tile, so the initially computed set S is large, and the more expensive box intersection test will be performed on many pairs. Overmars solves this problem by partitioning the set of cells into groups of similar size, and creating one hash table for each group. For collision detection, however, any one of the boxes may need to be checked for intersection with any other—this is not true in the point location problem—and so partitioning boxes into disjoint groups is not helpful. The solution is to build a hierarchical hash table, comprising several resolutions, and checking for intersections among boxes at different resolutions.

To understand the method, consider the one dimensional example, shown in Figure 2.11. Here, the six "boxes" (that is, line segments) populating space are labeled A through F. Let X denote an arbitrary box from this set, and define $\mathrm{sz}(X)$ as the size (in this case, length) of X. As a preprocessing step, one chooses constants α and β, and a minimal sequence of tiling resolutions, $\rho_1, \ldots, \rho_n$, such that

$$0 < \alpha < 1$$

$$\beta \geq 1$$

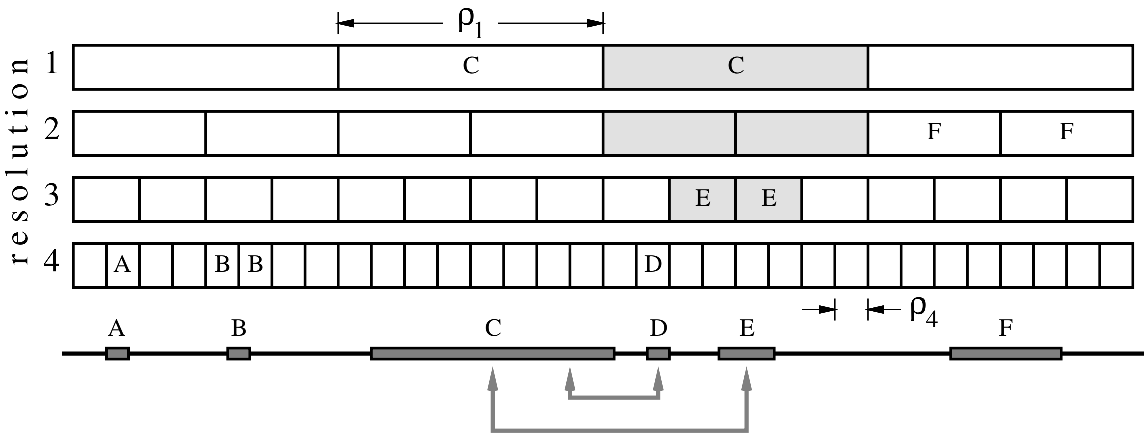

Figure 2.11: *A one-dimensional example of a hierarchical spatial hash table, with four resolutions. The value ρ_i is the size of the tiles at resolution i. The cells which must be checked when box E is stored in the table are shaded.*

$$\rho_1 > \rho_2 > \ldots > \rho_n > 0,$$

and so that for each box X there exists an integer $1 \leq k \leq n$ with

$$\alpha \leq \frac{\mathrm{sz}(X)}{\rho_k} \leq \beta. \tag{2.8}$$

The integer k is called the *resolution* of box X, abbreviated $\mathrm{res}(X)$. In Figure 2.11, $\alpha = 0.5$ and $\beta = 1.0$. This means that each box X must have a length that is from 0.5 to 1.0 times the width of the cells at resolution $\mathrm{res}(X)$. The constraints (2.8) are met by choosing four tiling resolutions, as shown in the figure, with $\mathrm{res}(A) = \mathrm{res}(B) = \mathrm{res}(D) = 4$, $\mathrm{res}(E) = 3$, $\mathrm{res}(F) = 2$, and $\mathrm{res}(C) = 1$. The location of box X is hashed at tiling resolution $\mathrm{res}(X)$. In two (or three) dimensions, the idea is the same. The cells are squares (or cubes) of side length ρ_i, and the boxes to be stored are rectangles (or rectangular prisms). For box X, $\mathrm{sz}(X)$ is the maximum distance between two opposite edges (or faces) of the box. In what follows, d denotes the dimension of the boxes and ambient space.

When a box is stored in the hash table, overlap with other boxes must be checked, some of which may be stored at other resolutions. Assume the boxes are hashed in order of increasing resolution. When box X is hashed, all enclosing cells at resolutions less than or equal to $\mathrm{res}(X)$ must be checked for other boxes. In Figure 2.11, the cells that must be checked when box E is stored are shaded. Since body C overlaps one of these cells, the bodies E and C are reported as *close*, meaning the hash table is unable to verify that the boxes do not overlap. Formally,

Definition 1 *Boxes X and Y are* **close** *if and only if they overlap a common cell at resolution* $\min(res(X), res(Y))$.

When boxes are close, full collision detection must be performed between the corresponding bodies. The storage requirements and relevant time complexities for a hierarchical spatial hash table scheme are now analyzed, assuming perfect hashing.

Lemma 1 *Let $sz_{\min}$ and $sz_{\max}$ be the sizes of the smallest and largest boxes to be stored in the hierarchical hash table. If n is the the number of resolutions required,*

$$n \leq \left\lceil \log_{\frac{\beta}{\alpha}} \frac{sz_{\max}}{sz_{\min}} \right\rceil.$$

Proof: Choose $\rho_1 = \mathrm{sz}_{\min}/\alpha$ and subsequent tile resolutions such that the constraint

$$\beta\rho_{i-1} = \alpha\rho_i \tag{2.9}$$

is satisfied for $2 \leq i \leq n$. In this way, an appropriate k can be found to satisfy (2.8) for any box dimension in the interval $[\mathrm{sz}_{\min}, \beta\rho_n]$; an n is needed such that $\rho_n \geq \mathrm{sz}_{\max}/\beta$. From (2.9),

$$\rho_n = \left(\frac{\beta}{\alpha}\right)^{n-1} \rho_1.$$

Substituting $\mathrm{sz}_{\min}/\alpha$ for ρ_1, and using $\rho_n \geq \mathrm{sz}_{\max}/\beta$, yields

$$\left(\frac{\beta}{\alpha}\right)^{n-1} \frac{\mathrm{sz}_{\min}}{\alpha} \geq \frac{\mathrm{sz}_{\max}}{\beta}.$$

The lemma follows. □

Lemma 1 gives an important theoretical bound, but it is not always tight. For instance, if all boxes are one of three sizes (small, medium, or large), than at most three resolutions are required for the hierarchical spatial hash table, regardless of the ratio of the dimensions of the largest to smallest box.

Theorem 3 *For a set of boxes to be stored in a hierarchical spatial hash table, let R be the ratio of the largest to smallest box dimension. Then the total number of hash buckets which must be checked for other boxes when storing a box in the hash table is $O(\beta^d \log R)$.*

Proof: When box X is stored, cells at resolutions $i \leq k$, where $k = \mathrm{res}(X)$, must be checked for other boxes. By (2.8), $\mathrm{sz}(X) \leq \beta\rho_k$, and so box X overlaps at most $(\beta + 1)^d$ cells at resolution k. The number of cells overlapped at a given resolution $i < k$ can not be more than this, and since there are $O(\log R)$ resolutions by Lemma 1, the number of overlapped cells is $O(\beta^d \log R)$. Each cell check corresponds to one bucket check in the hash table, and the theorem follows. □

Theorem 4 *Treating α, β, and R as constants, a hierarchical hash table can report all pairs of close boxes among n boxes in $O(n + c)$ time, where c is the number of close pairs.*

Proof: By Theorem 3, a constant number of hash buckets must be examined upon storing each box; the total number of buckets checked is $O(n)$. The time spent reporting closest pairs among these buckets is $O(c)$. □

A final theorem relates the resolution power of the hierarchical hash table to the parameter α.

Theorem 5 *The hierarchical spatial hash table can guarantee that two boxes X and Y do not intersect if the distance between them exceeds*

$$\frac{1}{\alpha}\sqrt{d}\max[sz(X), sz(Y)] \tag{2.10}$$

Proof: Without loss of generality, assume $\mathrm{sz}(X) \geq \mathrm{sz}(Y)$, so that $\mathrm{res}(X) \leq \mathrm{res}(Y)$. Assume X and Y are reported as close. Then they overlap a common cell at resolution $\mathrm{res}(X)$. The maximum distance between any two points in this cell is $\sqrt{d}\rho_{\mathrm{res}X}$, and so the distance d between boxes X and Y satisfies

$$d < \frac{\sqrt{d}\rho_{\mathrm{res}(X)}}{\mathrm{sz}(X)}\,\mathrm{sz}(X). \tag{2.11}$$

From (2.8),

$$\frac{\rho_{\mathrm{res}(X)}}{\mathrm{sz}(X)} \leq \frac{1}{\alpha}, \tag{2.12}$$

and the theorem follows. □

The tradeoffs involved in choosing the parameters α and β are now apparent. Recall that $0 < \alpha < 1$ and $\beta \geq 1$. By Theorem 3, the closer β is to the minimum value of 1, the fewer cells must be checked when storing boxes. By Theorem 5, the closer α is to the upper bound 1, the better the resolving power of the hash table. Finally, Lemma 1 implies that the larger the ratio β/α, the fewer resolutions are required to store all of the boxes.

2.4.2 Coherence and the tiling scheme

In simulation applications, a slight variant of the hierarchical hash table scheme can improve performance by taking advantage of coherence between problem instances. Instead of storing box X's label in buckets at $\mathrm{res}(X)$, and only checking the appropriate buckets at lower resolutions, the label is stored in all buckets that are checked. Using the example of Figure 2.11, the labels are stored as shown in Figure 2.12. Boxes X and Y are

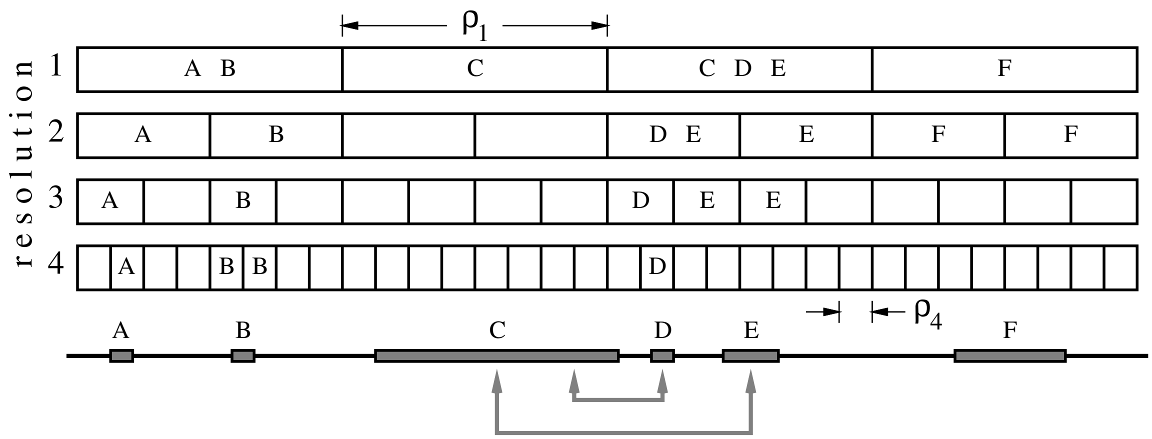

Figure 2.12: *In a variant of the hierarchical hash table scheme, the labels of each box are stored in every bucket that is checked for other boxes.*

reported as close if and only if their labels appear in a common hash bucket at resolution $\min(\mathrm{res}(X), \mathrm{res}(Y))$; the close pairs returned by the algorithm are exactly the same as with the original version. A two-dimensional array of *close counters* tracks which boxes are close. Each time X is stored into a hash bucket at resolution i that already contains Y, if $i = \min(\mathrm{res}(X), \mathrm{res}(Y))$, then the counter corresponding to the pair (X, Y) is incremented. When X is removed from such a bucket, the counter is decremented. Pairs of boxes for which the corresponding counter is zero are not close, and narrow phase collision checking is not performed between the corresponding bodies. When a counter is incremented from zero to one, the pair enters the set of bodies on which narrow phase collision checking is performed.

This scheme retains the state of the boxes between invocations. If a box intersects the same cells that it did on the last call, no additional work needs to be done to store that box on the current call. If the box has moved into new cells or left old ones, only buckets corresponding to these cells must be changed. Boxes corresponding to fixed bodies need only be stored into the hash table once. The disadvantage of this scheme is that more processing is sometimes required to store a box's label into a bucket. In Figure 2.12, even though D and E are verified as not close at resolution 3, they must be stored in common buckets at resolutions 2 and 1, requiring extra processing and adjustment of close counters. For this reason, the claim of Theorem 4 is not valid (or at least not readily apparent) for this variant on the algorithm. However this variant is quite efficient in practice, as shown in Table 2.1. The table shows the number of cycles spent on broad phase collision detection for

	millions of cycles		
example	standard hashing	coherence hashing	ratio
coins	361	58	6.2
bowling	2188	314	7.0
rattleback top	641	107	6.0
part feeder chute	409	57	7.2

Table 2.1: *Comparison of hashing schemes.*

some example simulations, using both the standard hashing algorithm and the coherence hashing algorithm. The simulations themselves are described in detail in Chapter 7. From the table, the coherence hashing algorithm is significantly better, consistently running six or more times faster than the standard hashing algorithm. For this reason, the coherence hashing algorithm is used in *Impulse*.

2.4.3 Maintaining the collision heap

The hierarchical spatial hash table described in the last section can be used to cull unnecessary collision checks and TOI updates for a group of moving bodies in a simulation. Upon each call to the integrator, the state of the system is evolved from the current time t_0 to some future time $t_0 + \Delta t$. The step Δt is determined by the earliest TOI field on the top of the collision heap. After Δt is computed, but before the integrator is called, for each body an axes aligned bounding box is computed that is guaranteed to enclose the swept volume of the body during the upcoming integration.

Consider ballistic body i, with bounding box B_i. The center of mass of body i follows a parabolic trajectory that is known from the current state of the body. B_i is found by noting the position of body i's center of mass at the current time t_0, at the time $t + \Delta t$, and possibly at the apex of its parabolic trajectory, should this occur during the interval $[t_0, t + \Delta t]$. The box which bounds these two or three points is grown by r_i, the maximum radius of body i, to obtain B_i (Figure 2.13). Clearly, computing B_i involves a fixed number

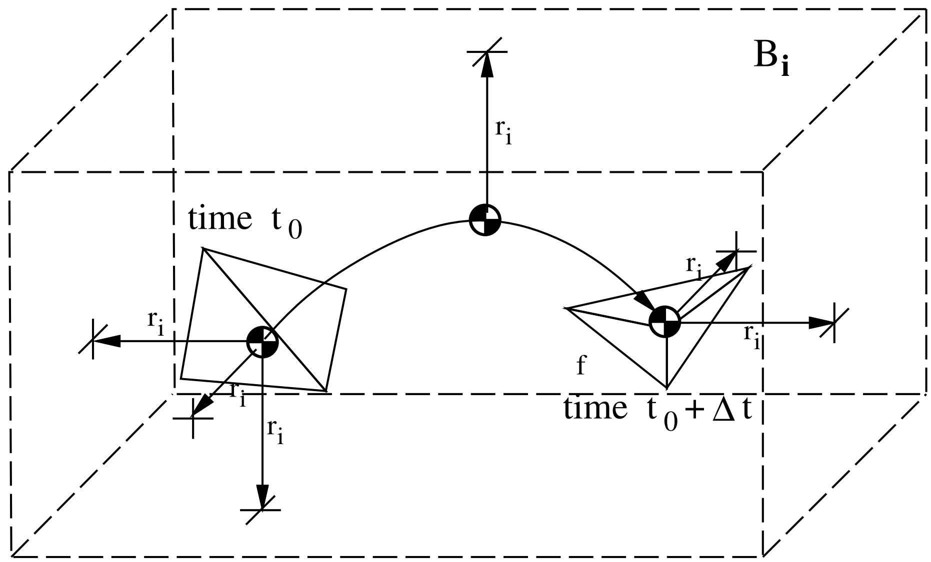

Figure 2.13: *The bounding box for body i's swept volume during a segment of a ballistic trajectory.*

of operations, and can be done in constant time. This fact and Theorem 4 imply that all bounding boxes can be computed, stored in the hash table, and close pairs reported, all in $O(n + c)$ time for an n-body simulation (c is the number of close boxes). This is done at the beginning of each integration step. For static bodies such as walls or platforms, it is not necessary to grow the bounding box by the body radius, and the resulting bounding boxes are much tighter; also, boxes for fixed bodies need never be updated.

For each pair of bodies that the hash table deems close, there is a corresponding element maintained in the collision heap. Pairs not deemed close are in no danger of colliding during the next integration step. When the close counter for a particular pair is incremented from zero to one, the TOI for that pair is computed, and the pair is added to the heap, if it is not there already. Since pairs in the heap are kept sorted on the TOI field, no pair can collide any sooner than the pair at the top of the heap. The `advance` algorithm (Figure 2.14) is performed at the current integration step. As can be seen from the algorithm, new pairs are inserted into the active collision heap before any integration occurs. This is because the computed TOI for such a pair might precede the TOI previously

```
advance

    Δt ← TOI at top of heap − t_0
    for each body i
        compute bounding box B_i over interval [t_0, t_0 + Δt]
        store B_i in hierarchical hash table
        for each j such that closeCounts(i, j) was incremented to 1
            if pair (i, j) is not in heap
                compute TOI between bodies i and j
                store pair in collision heap
    /* since top TOI might have decreased due to new pairs...  */
    Δt ← TOI at top of heap − t_0
    integrate state of system over [t_0, t_0 + Δt]
```

Figure 2.14: `advance`. Advance the state of the system forward in time, as far as possible, while guaranteeing no collisions are missed.

at the top of the heap, in which case the integration step must be shortened.

The question of when pairs should be removed from the heap remains. One approach would be to remove pairs as soon as the hash table indicates the bodies' boxes are no longer close. This can cause pairs of bodies to rapidly move in and out of the heap, especially when one of the boxes is close to the boundary of a tiling cell (Figure 2.15). To avoid this inefficient behavior, hysteresis is applied to the collision heap. A body pair is added to the heap whenever the boxes become close, but the pair may not be removed before it bubbles up to the top of the heap. After a collision check for the top heap element is made, the closest counter status for that pair is examined. If it is nonzero, the boxes are still close, and the pair is reinserted into the heap after computing the new TOI. If the closest counter for that pair is zero, the pair leaves the heap.

The hierarchical hashing scheme also reduces TOI updates upon collision. Recall that without the hashing scheme, whenever bodies i and j collided, all TOIs involving either one of these bodies must be recomputed. However, with the hashing scheme, TOIs

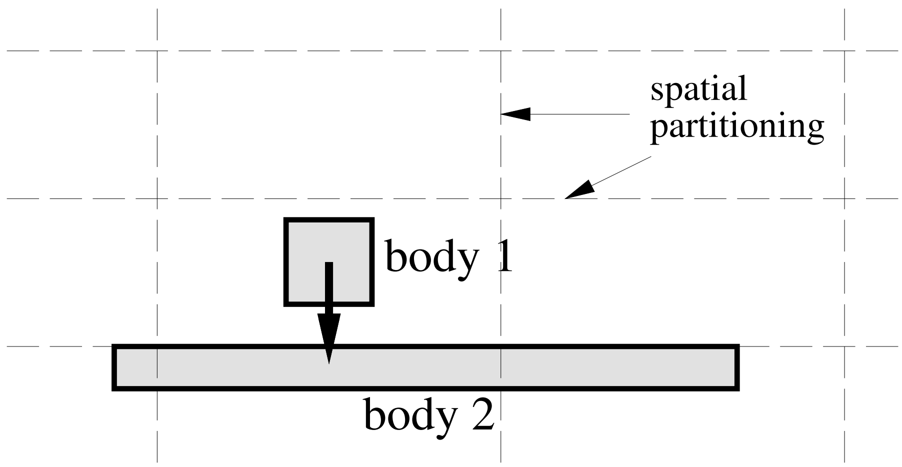

Figure 2.15: *If body 1 is bouncing on fixed body 2, the pair of boxes corresponding to these bodies may rapidly toggle between close and not close status. To avoid this jittering, hysteresis is applied to the collision heap.*

are only maintained for pairs in the heap. As a result, if bodies i and j collide, TOIs are only recomputed between these bodies and nearby bodies. This greatly reduces the cost of processing collisions.

The sizes of the bounding boxes depend on the current (linear) velocities of the bodies. These can not be known for all time at the beginning of the simulation. As a result, tiling resolutions can not be chosen as described in Lemma 1. In practice, this is not such a problem. In *Impulse*, the tiling resolutions are based simply upon the maximum radii of the bodies. Unless the bodies are moving at an extremely fast speed, the number of tiles intersected by the various boxes remains small. Figure 2.16 shows the reduction in narrow phase collision detection due to the hierarchical hashing scheme.

2.4.4 Spatial hashing versus coordinate sorting

In addition to the hashing scheme described above, there is another algorithm for finding intersections of axes-aligned bounding boxes. The algorithm is based on sorting the coordinates of edges of the bounding boxes along each of the three coordinate axes; it is used in the *I-COLLIDE* system [CLMP95] and also in [Bar92]. The algorithm works as follows. The minimum and maximum x-coordinates of each axes-aligned box are maintained in a sorted list. The same is done for the y and z coordinates. Two boxes overlap if and only if their coordinates overlap in each of the three coordinate directions. A two-dimensional

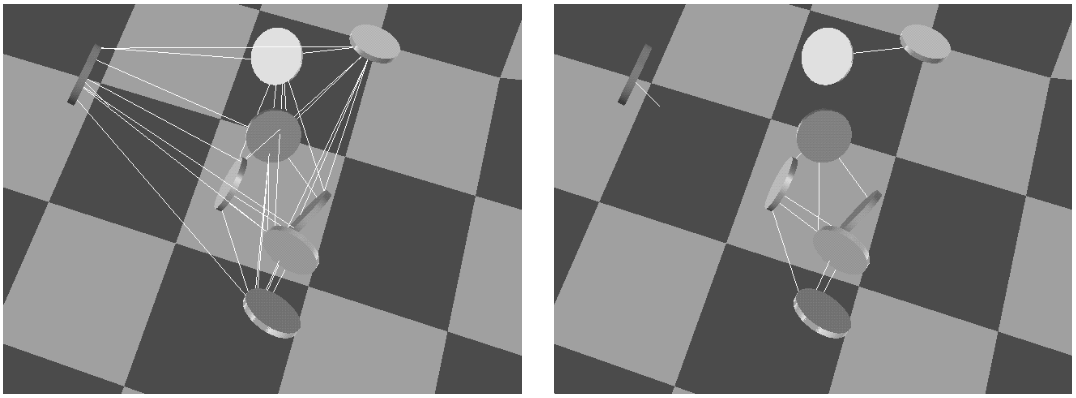

Figure 2.16: *Snapshots taken during the simulation of eight coins tossed onto a flat surface. The lines indicate the tracking of closest points between bodies; each line corresponds to one element in the active collision heap. The left figure was produced with the broad phase collision detection deactivated, and the right one with the broad phase activated. The broad phase greatly culls collision checks. Also, when a body experiences a collision impulse, only the TOIs with bodies connected to it by lines need be updated.*

example is shown in Figure 2.17. For example, since x_1', the maximum x-coordinate of box B_1 is less than x_2, the minimum x-coordinate of box B_2, these boxes can not overlap. On the other hand, $x_3 < x_1' < x_3'$, and $y_1 < y_3 < y_1'$. Thus, boxes B_1 and B_3 overlap in both the x and y coordinates, and therefore the boxes themselves overlap. Cohen, *et. al.* discuss the relative merits of using a fixed size, cubical box that can accommodate a body at any orientation versus tighter fitting boxes that change in shape as the body rotates [CLMP95].

Coherence is exploited by updating previously sorted lists to obtain new sorted lists. In this way, the number of exchanges needed to obtain the new sorted list is expected to be $O(n)$. It can, however, be $O(n^2)$. Consider the situation depicted in Figure 2.18. The maximum and minimum y-coordinates of all the boxes are clustered closely together. Even with very small motions from one time step to the next, $O(n^2)$ exchanges result in resorting the y-coordinates; coherence breaks down. This example is not contrived. Imagine throwing a group of dice onto a flat horizontal surface. As the dice come to rest, their bounding boxes will tend to cluster along the vertical coordinate. Since coordinate sorting is based on dimension reduction, the coordinates may be clustered even when the original boxes are not; the clustering becomes worse in higher dimensions. One way of handling the clustering problem is to perform a less drastic dimension reduction, projecting the three-dimensional boxes first into two-dimensional rectangles in the plane, and reporting intersections among

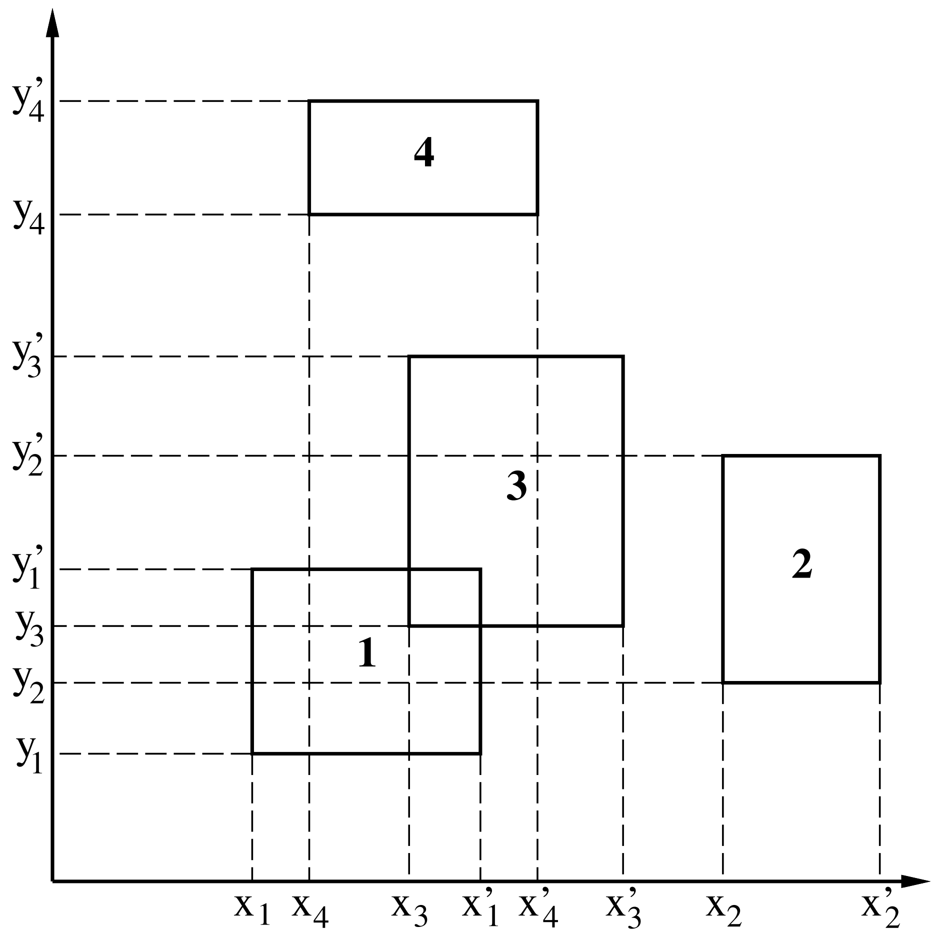

Figure 2.17: *A two dimensional version of* I-COLLIDE's *bounding box check. Two boxes overlap if and only if their projections onto the x- and y-axes overlap.*

the rectangles in $O(n \log n + k)$ time, where k is the number of intersections [Ede83]. Hashing schemes do not suffer from the clustering problem. Coherence always results in efficient updating of the hash table, unless the number of box overlaps in *three* dimensions is large.

Coordinate sorting does have one advantage over hashing: no hashing scheme culls as many body pairs as coordinate sorting. Cohen, *et. al.* claim that choosing a near-optimal cell size is difficult, and failing to do so results in large memory usage and computational inefficiency. These claims are largely mitigated with a hierarchical hash table based on multiple cell sizes. A very efficient collision detection scheme based on boxes that are *not* axes aligned is described in [GLM96].

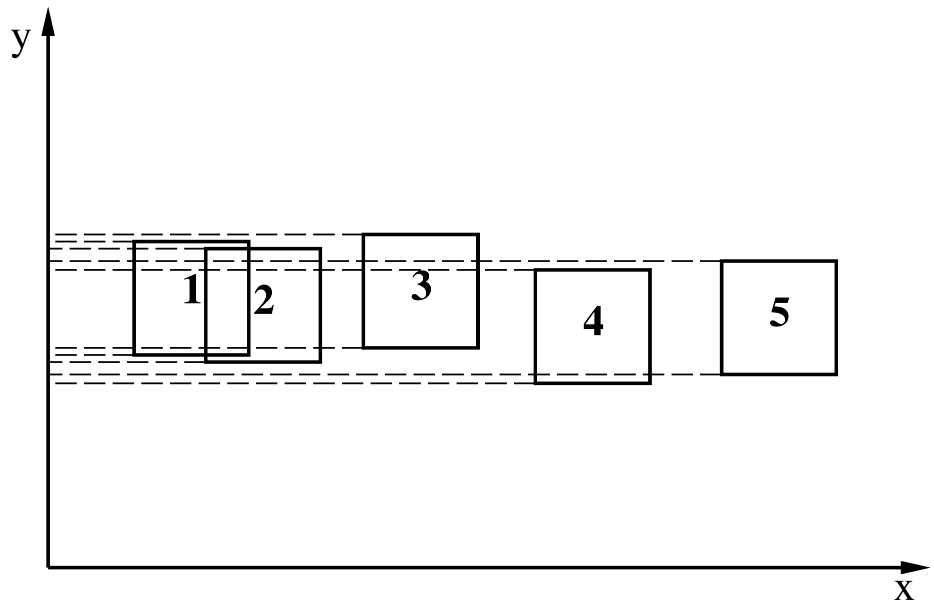

Figure 2.18: *A bad case for coordinate sorting. The dense clustering of box extrema along the y-axis results in $O(n^2)$ exchanges for each new sort of the coordinates.*

2.5 Generalizing collision detection

The collision detection techniques described thus far are tailored to ballistic bodies, which are ubiquitous in impulse-based simulation. The techniques are extensible to other types of motion with the definition of two routines:

1. A **swept volume** routine. This takes the state of the body at the current time t_0, and a time interval Δt, and returns an axes-aligned box that encloses the center of mass's trajectory during the time interval $[t_0, t_0 + \Delta t]$.

2. A **time of impact coefficients** routine. This takes the current state of the body, and a directional vector $\hat{\mathbf{d}}$. It returns two coefficients, A and B, such that the distance any point on the body travels in the direction of $\hat{\mathbf{d}}$ is bounded by the expression

$$A(t - t_0)^2 + B(t - t_0), \quad t > t_0. \tag{2.13}$$

The swept volume routine is needed during the broad phase of collision detection and the TOI coefficient routine is needed during the narrow phase. Previous sections described how the swept volume and TOI coefficients are estimated for ballistic bodies. For fixed bodies, the routines are trivial: the swept volume is always the current volume occupied by the body, and the TOI coefficients are both zero. The swept volume and TOI coefficient routines for bodies connected by joints are discussed in Chapter 5.

Another type of motion is *scripted* motion. Scripted bodies follow unalterable trajectories through state space; they are impervious to external forces like gravity, and collisions with other bodies. An example application of a scripted body is a vibrating part feeder. Since the mass of the feeder is much greater than the mass of the small parts it vibrates along a track, one could neglect the effects on the feeder of collisions with the parts. Instead, the feeder might execute some prescribed, sinusoidal motion. *Impulse* provides several types of scripted bodies, and others could be added for specific applications. Some simulations with scripted bodies are described in Chapter 7. One type of scripted body is a *cycler*. A cycler's center of mass follows an elliptical path through space, while the cycler's orientation remains constant. For illustration, consider the case where the path is a circle in the x-y plane. Let r be the radius of this circle, and let Ω be the angular speed at which the center of mass moves around the circle. This is not the same as the angular velocity of the body, which is zero since the orientation is held constant.

First consider the swept volume routine. Since the center of mass remains in the x-y plane, the minimum and maximum z-coordinates of the center of mass are both 0. The minimum and maximum x- and y-coordinates are inferred from the smallest axes aligned rectangle in the x-y plane which encloses several points: $\mathbf{p}(t_0)$, the current location of the center of mass; $\mathbf{p}(t_0 + \Delta t)$, the location of the center of mass at a time Δt in the future; and any crossings of the trajectory with the x- and y-axes over the interval $[t_0, t_0 + \Delta t]$ (Figure 2.19). All of these points are easily computed since the trajectory in the x-y plane

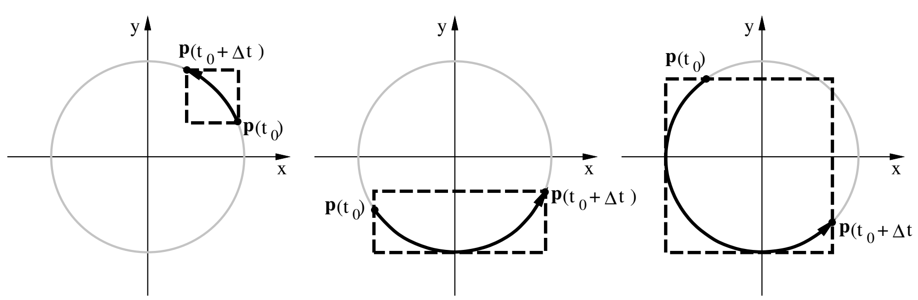

Figure 2.19: *The axes aligned bounding rectangles in the x-y plane for a cycler body. Three different cases are shown. The point $\mathbf{p}(t_0)$ is the current position of the center of mass, and the point $\mathbf{p}(t_0 + \Delta t)$ is the position of the center of mass a time Δt in the future.*

is known in closed form:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos\Omega t \\ r\sin\Omega t \end{bmatrix}.$$

Now consider the TOI coefficients for this cycler. Since the angular velocity of a cycler is zero, every point on the body has the same velocity as the center of mass at all times. Thus it suffices to bound the motion of the center of mass along the directional vector $\hat{\mathbf{d}}$. There are different ways to do this. Let $\mathbf{v}(t)$ be the linear velocity of the center of mass of the body. The magnitude of $\mathbf{v}(t)$ is a constant, equal to $r\Omega$. Therefore, if D is the distance that any point on the body travels in the direction $\hat{\mathbf{d}}$,

$$D \le r\Omega(t - t_0). \tag{2.14}$$

This bound suggests choosing the TOI coefficients in (2.13) to be $A = 0$ and $B = r\Omega$.

A second method for bounding motion is to observe that the current velocity of the center of mass in the direction $\hat{\mathbf{d}}$ is given by $\mathbf{v}(t_0) \cdot \hat{\mathbf{d}}$. The magnitude of the acceleration of the center of mass is bounded by $r\Omega^2$. This is the centripetal acceleration, always directed toward the center of rotation. Thus,

$$D \le \frac{1}{2} r\Omega^2 (t - t_0)^2 + [\mathbf{v}(t_0) \cdot \hat{\mathbf{d}}](t - t_0). \tag{2.15}$$

This suggests choosing the TOI coefficients $A = \frac{1}{2} r\Omega^2$ and $B = \mathbf{v}(t_0) \cdot \hat{\mathbf{d}}$.

The coefficients indicated by (2.14) and (2.15) are for a linear and quadratic model for distance traveled, respectively. In general, the quadratic model has a smaller linear term and will be a tighter bound on the distance traveled up to some time. After that time, the quadratic term will dominate, and the quadratic bound will overtake the linear one. To decide which model to apply, one can use the current distance d to the other body. *Impulse* checks the distance at which the quadratic model overtakes the linear one. If this is greater than d, the quadratic coefficients are returned, otherwise the linear coefficients are returned. The time at which the two models are equal is obtained by equating the right hand sides of (2.14) and (2.15):

$$(r\Omega)(t - t0) = \frac{1}{2} r\Omega^2 (t - t_0)^2 + [\mathbf{v}(t_0) \cdot \hat{\mathbf{d}}](t - t_0),$$

which yields

$$t - t_0 = \frac{2\left[r\Omega - \mathbf{v}(t_0) \cdot \hat{\mathbf{d}}\right]}{r\Omega^2}.$$

At this time, the distance bound given by both models is

$$D = \frac{2\left[r\Omega - \mathbf{v}(t_0) \cdot \hat{\mathbf{d}}\right]}{\Omega}.$$

If this bound is less than the distance to the other body, the cycler TOI coefficients routine returns the coefficients of t^2 and t in (2.14), otherwise the coefficients in (2.15) are returned. This strategy gives whichever bound is likely to be tighter.

Other types of motion may be handled in the same way; the analyses are similar to the one for cyclers described above. It is not necessary to perform analyses based on *pairs* of motion types, for instance for cycler-ballistic pairs, cycler-fixed pairs, ballistic-fixed pairs, and so on. Rather, the swept volume and TOI coefficient routines are defined for each single motion type, and then collision detection can be performed between this type and all other motion types. For n types of motion, only $O(n)$ analyses are required instead of $O(n^2)$. This feature is also shared by the collision detection algorithm of Von Herzen, *et. al.* [HBZ90]. Lipschitz bounds need only be computed on a per body basis, not on a per body pair basis. The bounds for individual types are combined to form the bound for a particular pair.

Chapter 3

Collision Response

The collision response problem is concerned with computing the pair of equal and opposite impulses that should be applied to the colliding bodies in order to prevent penetration. Like contact forces, collision impulses are subject to frictional constraints, as well as other constraints governing the energy dissipation during collisions. Collision response algorithms fall into two broad categories. The first category makes the *constant sliding direction* assumption: the direction of the relative tangential velocity between colliding bodies at the contact point remains constant during a collision. This assumption is convenient because the direction of the frictional force varies with the sliding direction. Holding these directions fixed allows one to solve for the collision impulse and post-collision velocities of the bodies by solving a system of algebraic equations. Many variations on this theme are described by Brach [Bra91]. In the context of interactive dynamic simulation, this approach is used almost exclusively [Bar92, CS89, Hah88, LRK94, MW88, NM93]. It is often a fairly strong approximation, particularly for collisions between three-dimensional bodies.

Just as a collision impulse changes the relative normal velocity between colliding bodies, it also changes the relative tangential velocities; the direction of the latter is not constant. The second class of collision response methods use differential equations that describe the collision process. They account for the change in sliding direction and the possibility of transient sticking that can occur during impact. These approaches date back to Routh [Rou05], and have been studied by many others [BK94, Kel86, MC95a, Str91, WM87]. For impulse-based simulation, an accurate collision model is required for physically valid results.

This chapter describes a suitable collision resolution algorithm for an impulse-based simulator, which has been implemented in *Impulse*. The main contribution is a computational model for the differential analysis of the collision process; this approach is more accurate than the algebraic one. Building on work of Routh and Keller, new equations are derived that are suitable for numeric integration. The algorithm presented here also incorporates Strong's improved model for restitution. To accomplish this, the collision integration is divided into three phases, and two different parameterizations are used. This chapter also presents new results concerning the determinacy of collision dynamics under the Coulomb friction model. Recently, Bhatt and Koechling have explored these issues [BK94, BK96b], and in [BK96a] they report some of the same results. The geometric proofs given in this chapter differ significantly from theirs, which are based on the roots of polynomials; in our opinion, the geometric proofs are simpler and more intuitive. Finally, this chapter discusses some approaches for modeling static friction in an impulse-based context.

3.1 Assumptions of collision response model

The physical phenomena that occur between bodies in contact result from complex interactions at the atomic level. Finite element methods are a first step toward a tractable model, but are still too computationally expensive to be used in the context of interactive simulation A much greater approximation, but one which significantly reduces computational costs is the rigid body model. This model is reasonable for many everyday physical systems; it is most tenuous during collisions between bodies, which involve surface deformations. Still, many physical phenomena can be captured with a rigid body model, it has wide application in engineering, and it is the model used by *Impulse*.

Assumption 1 (Rigid bodies) *All physical objects in the simulation are perfectly rigid.*

This model has several implications. When two objects collide, the duration of the collision is infinitesimal, and the forces needed to prevent penetration are impulsive, instantaneously changing the velocities of the colliding bodies. Since velocities are finite, the positions of the colliding bodies are constant during the collision. Non-impulsive forces (like gravity) have no effect over infinitesimal intervals, and may thus be ignored in collision analysis. The rigid body assumption is made frequently in collision analysis.

When two real bodies collide, there is a period of deformation during which elastic energy is stored in the bodies, followed by a period of restitution during which some of this energy is returned as kinetic energy, as the bodies rebound off each other (Figure 3.1). The

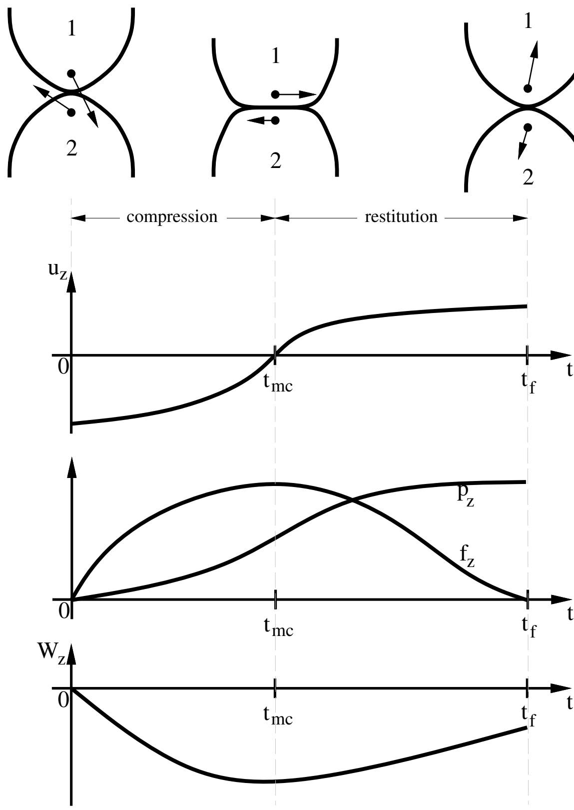

Figure 3.1: *The normal velocity, force, impulse, and work during the compression and restitution phases of a collision. The 'z' subscript denotes the normal direction at the collision point.*

rigid body assumption is usually coupled with a model to approximate this process. The simplest model, and the one used in most elementary physics texts, is Newton's impact law:

$$u_z(t_f) = -e\ u_z(0).$$

Here u_z is the normal component of the relative velocity between the bodies; $u_z(t) < 0$, means the bodies are moving toward another at time t, and $u_z(t) > 0$ means they are separating. The collision starts at time zero, and ends at time t_f. The point of maximum compression, t_{mc}, is the point at which the normal velocity changes sign, and the phase changes from compression to restitution. The constant e is called the *coefficient of restitution*, and make take values between 0 and 1, depending on material properties of the colliding bodies. Newton's law is linear, and lends itself nicely to an algebraic solution for the post-collision velocities.

Poisson's hypothesis, an alternative restitution law, uses e as a ratio between impulses rather than velocities:

$$p_z(t_f) - p_z(t_{\mathrm{mc}}) = e\ p_z(t_{\mathrm{mc}}).$$

Here, p_z is the normal impulse, the time integral of the normal force f_z delivered from one body to another. Poisson's hypothesis states that the normal component of impulse delivered during the restitution phase is e times the normal component of impulse delivered during the compression phase. This is equivalent to Newton's law for frictionless collisions. Keller uses Poisson's hypothesis to derive equations of collision with friction [Kel86].

Both Newton's impact law and Poisson's hypothesis can cause the total energy of the colliding bodies to increase during a collision, when friction is present. To correct this defect, Stronge proposes a new definition of the coefficient of restitution, as a ratio of the work done by the normal components of impulse [Str91]:

Assumption 2 (Stronge's hypothesis) *Let $W_z(t)$ be the work done by the normal components of the (equal and opposite) collision impulses during a collision. Then*

$$W_z(t_f) - W_z(t_{mc}) = -e^2\ W_z(t_{mc}).$$

The positive work done during the restitution phase is $-e^2$ times the negative work done during compression. Unlike the other models, Stronge's model guarantees that the effects of the normal forces, like the tangential frictional forces, are always dissipative; they can not add energy to the system. This agrees with the description of the physical process of collision. Stronge's model is employed by the simulator *Impulse*.

The tangential components of fore and impulse that develop during a collision are governed by a friction law. A very common friction formulation, and the one employed in *Impulse*, is the Coulomb friction law:

Assumption 3 (Coulomb friction law) *At some instant during a collision between bodies 1 and 2, let $\mathbf{u}$ be the contact point velocity of body 1 relative to the contact point velocity of body 2. Let $\mathbf{u}_t$ be the tangential component of $\mathbf{u}$, and let $\hat{\mathbf{u}}_t$ be a unit vector in the direction of $\mathbf{u}_t$. Let $\mathbf{f}_z$ and $\mathbf{f}_t$ be the normal and tangential (frictional) components of force* *exerted by body 2 on body 1, respectively. Then*

$$\mathbf{u}_t \neq \mathbf{0} \quad \Rightarrow \quad \mathbf{f}_t = -\mu\|\mathbf{f}_n\|\hat{\mathbf{u}}_t$$
$$\mathbf{u}_t = \mathbf{0} \quad \Rightarrow \quad \|\mathbf{f}_t\| \leq \mu\|\mathbf{f}_n\|$$

where μ is the coefficient of friction.

In order, the equations above correspond to dynamic and static friction. While the bodies are sliding relative to one another, the frictional force is exactly opposed to the direction of sliding. If the relative tangential velocity is zero, all that is known is that the total force lies in a friction cone.

3.2 Computing collision impulses

Collision processing is initiated when two criteria are met:

1. The collision detection system reports that the distance between two bodies is less than the collision epsilon.
2. The velocities of these bodies are such that the distance between the closest points is decreasing.

The job of the collision resolution system is to compute a pair of equal and opposite collision impulses to be applied to the colliding bodies at the contact points. These impulses must prevent penetration of the bodies, and also satisfy the physical laws discussed in the previous section. This section describes a method for computing these impulses. The situation is as depicted in Figure 3.2. The collision frame $\mathcal{F}_{\text{coll}}$ has its origin at the collision point and its z-axis aligned with the mutual surface normal at this point, pointing from body 2 toward body body 1. For non-smooth bodies, such as polyhedra, the surface normal is defined to lie along the line between the closest points. For body i: m_i is the mass of the body; $\mathbf{I}_i$ is the 3×3 mass matrix, described in Appendix A.3; $\mathbf{v}_i$ is the linear velocity of the enter of mass; $\boldsymbol{\omega}_i$ is the angular velocity of the body; $\mathbf{u}_i$ is the absolute velocity of the contact point on the body; and $\mathbf{r}_i$ is the offset vector from the body's center of mass to the contact point. Also, $\mathbf{p}$ is the collision impulse delivered from body 2 to body 1, and $-\mathbf{p}$ is the reaction impulse delivered from body 1 to body 2. Throughout this section, it is assumed that all vectors and tensors are resolved in the collision frame.

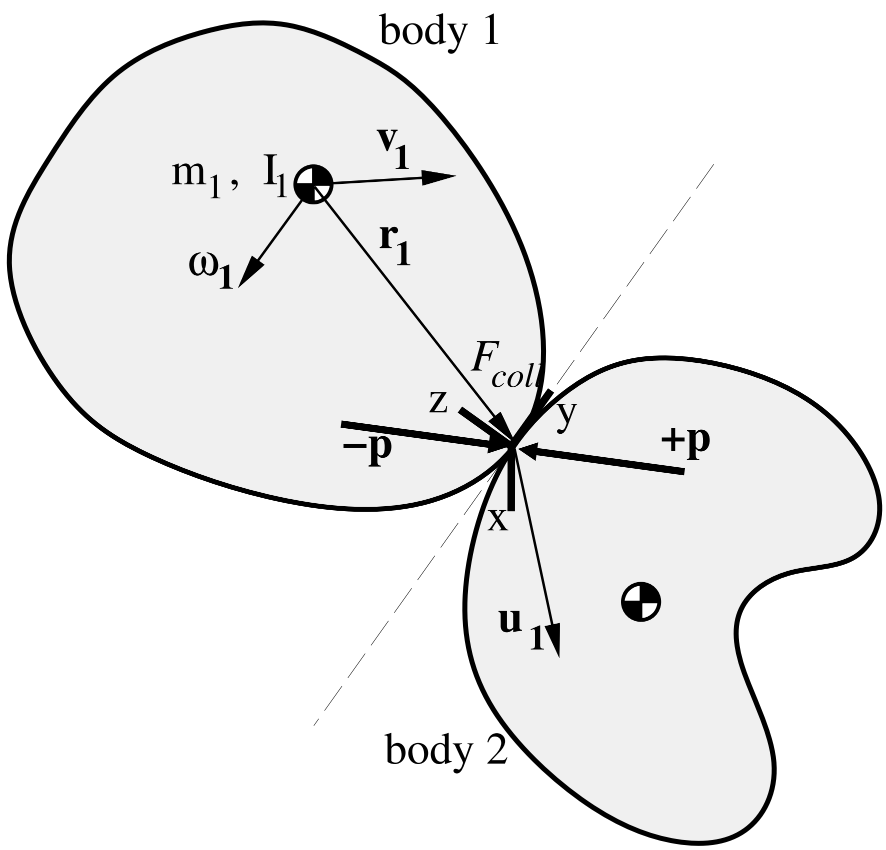

Figure 3.2: *A collision between two bodies, indicating some of the quantities used during collision analysis. The collision frame $\mathcal{F}_{coll}$ is chosen so that the z-axis is aligned with the surface normals at the collision point.*

3.2.1 The equations of collision

Basic Newtonian physics dictate how collision velocities and impulses evolve. During a collision, all non-impulsive forces acting on body 1 are negligible; only the collision force $\mathbf{f}(t)$ needs to be considered, which also induces a torque of $\mathbf{r} \times \mathbf{f}(t)$. From the Newton-Euler equations (Appendix A.3),

$$\begin{aligned} \mathbf{f}(t) &= m\mathbf{a}_1(t) \\ \mathbf{r}_1 \times \mathbf{f}(t) &= \mathbf{I}_1\boldsymbol{\alpha}_1(t) + \boldsymbol{\omega}_1(t) \times \mathbf{I}_1\boldsymbol{\omega}_1(t), \end{aligned}$$

where $\mathbf{a}_1$ and $\boldsymbol{\alpha}_1$ are the linear and angular accelerations of body 1. The last term in the second equation comprises inertial forces, which are also negligible during collision; it may be dropped. Also, $m_1, \mathbf{I}_1$, and $\mathbf{r_1}$ are constants. With these facts in mind, the above equations can be integrated over time to give

$$\begin{aligned} \mathbf{p}(t) &= m_1 \Delta \mathbf{v}_1(t) && (3.1) \\ \mathbf{r}_1 \times \mathbf{p}_1(t) &= \mathbf{I}_1 \Delta \boldsymbol{\omega}_1(t), && (3.2) \end{aligned}$$

where $\mathbf{p}$ is the collision impulse:

$$\mathbf{p}(t) = \int_0^t \mathbf{f}(\tau)\, d\tau. \qquad (3.3)$$

These equations relate the impulse delivered up to some time t to body 1's velocity changes up to that point. Using time as a parameter is valid for the above derivation, in which a collision is a event of finite duration, during which large but bounded forces act. For the rigid body model, however, a collision is really the limit of this process, as t_f approaches zero and the collision forces become infinite. Difficulties in analysis arise because the collision occurs over a zero time interval, and the velocities become discontinuous functions of time. These problems are remedied by choosing a new parameterization for the collision. A collision parameter γ is chosen, which monotonically increases during the course of the collision. All velocities of the colliding bodies as well as the accumulated impulse are expressed as functions of γ. One natural choice $\gamma = p_z$, the normal component of $\mathbf{p}$. Clearly p_z begins at zero and monotonically increases during a collision: it is the integral of the normal component of force exerted by body 2 on body 1, and this force component is always positive in $\mathcal{F}_{\text{coll}}$ (Figure 3.2) because the bodies can only push on each other. Furthermore, velocities will be shown to be continuous functions of this parameter during a collision. Other collision parameters will be used as well; when generality is required, γ denotes an arbitrary collision parameter. A prime ($\prime$) is sometimes used to denote differentiation with respect to the collision parameter.

Rewriting (3.1) and (3.2), and changing to the new collision parameter γ:

$$\begin{aligned} \Delta \mathbf{v}_1(\gamma) &= \frac{1}{m_1}\mathbf{p}(\gamma) \\ \Delta \boldsymbol{\omega}_1(\gamma) &= \mathbf{I}_1^{-1}\mathbf{r}_1 \times \mathbf{p}(t). \end{aligned}$$

The contact point velocity $\mathbf{u}_1$ is given by

$$\mathbf{u}_1(\gamma) = \mathbf{v}_1(\gamma) + \boldsymbol{\omega}_1(\gamma) \times \mathbf{r}_1.$$

Combining the above three equations,

$$\begin{aligned}\Delta\mathbf{u}_1(\gamma) &= \frac{1}{m_1}\mathbf{p}(\gamma) + \left(\mathbf{I}_1^{-1}\mathbf{r}_1 \times \mathbf{p}(t)\right) \times \mathbf{r}_1 \\ &= \left(\frac{1}{m_1}\mathbf{1} - \tilde{\mathbf{r}}_1\mathbf{I}_1^{-1}\tilde{\mathbf{r}}_1\right)\mathbf{p}(\gamma),\end{aligned}$$

where $\tilde{\mathbf{r}}_1$ is the cross-product matrix corresponding to $\mathbf{r}_1$ (Appendix A.2), and $\mathbf{1}$ is the 3×3 identity matrix. The same derivation can be performed for body 2; the only difference is that $\mathbf{p}$ is replaced with the reaction $-\mathbf{p}$. The result is:

$$\Delta\mathbf{u}_2(\gamma) = -\left(\frac{1}{m_2}\mathbf{1} - \tilde{\mathbf{r}}_2\mathbf{I}_2^{-1}\tilde{\mathbf{r}}_2\right)\mathbf{p}_2(\gamma).$$

Letting $\mathbf{u}(\gamma)$ denote the *relative* contact point velocity, that is $\mathbf{u}_1 - \mathbf{u}_2$,

$$\Delta\mathbf{u}(\gamma) = \underbrace{\left[\left(\frac{1}{m_1} + \frac{1}{m_2}\right)\mathbf{1} - \left(\tilde{\mathbf{r}}_1\mathbf{I}_1^{-1}\tilde{\mathbf{r}}_1 + \tilde{\mathbf{r}}_2\mathbf{I}_2^{-1}\tilde{\mathbf{r}}_2\right)\right]}_{\mathbf{K}}\mathbf{p}(\gamma), \tag{3.4}$$

or more succinctly,

$$\Delta\mathbf{u}(\gamma) = \mathbf{K}\mathbf{p}(\gamma). \tag{3.5}$$

The 3×3 matrix $\mathbf{K}$ is called the collision matrix; it plays a central role in computing the impulse to resolve the collision.

Theorem 6 (properties of K) *For a given collision, the collision matrix* $\mathbf{K}$ *defined by (3.4) is constant, nonsingular, symmetric, and positive definite.*

Proof: Constancy of $\mathbf{K}$ is evident from (3.4): it depends only on the masses, mass distributions, and contact point locations of the colliding bodies, all of which are constant during a collision. For the other claims, note that $\mathbf{K}$ is the sum of three matrices: a positive scalar multiple of the identity matrix, which is clearly symmetric positive definite; $\mathbf{A}_1 = -\tilde{\mathbf{r}}_1\mathbf{I}_1^{-1}\tilde{\mathbf{r}}_1$; and $\mathbf{A}_2 = -\tilde{\mathbf{r}}_2\mathbf{I}_2^{-1}\tilde{\mathbf{r}}_2$. The mass matrices $\mathbf{I}_i$ are also symmetric positive definite (see Chapter 6), therefore so are their inverses. Since $\tilde{\mathbf{r}}_i$ is skew-symmetric, it follows that $\mathbf{A}_i^T = \mathbf{A}_i$, so $\mathbf{A}_i$ is symmetric. For an arbitrary vector $\mathbf{x}$,

$$\mathbf{x}^T\mathbf{A}_i\mathbf{x} = -\mathbf{x}^T\tilde{\mathbf{r}}_i\mathbf{I}_i^{-1}\tilde{\mathbf{r}}_i\mathbf{x} = (\tilde{\mathbf{r}}_i\mathbf{x})^T\,\mathbf{I}_i^{-1}\,(\tilde{\mathbf{r}}_i\mathbf{x}) = \mathbf{w}^T\mathbf{I}_i^{-1}\mathbf{w},$$

where $\mathbf{w} = \mathbf{r}_i \times \mathbf{v}$. Since $\mathbf{I}_i^{-1}$ is positive definite, the right hand side is non-negative, and therefore $\mathbf{A}_i$ is positive semi-definite. Finally, since $\mathbf{K}$ is the sum of three symmetric matrices, of which one is positive definite and two are positive semi-definite, $\mathbf{K}$ is symmetric positive definite, which also implies nonsingularity. □

The main result of this section is embodied in the following theorem.

Theorem 7 (equations of collision) *During a collision parameterized by γ, the relative contact point velocity and the collision impulse evolve according to:*

$$\begin{aligned} \frac{d}{d\gamma}\mathbf{u}(\gamma) &= \mathbf{K}\frac{d}{d\gamma}\mathbf{p}(\gamma) \\ \frac{d}{d\gamma}\mathbf{p}(\gamma) &= \mathbf{K}^{-1}\frac{d}{d\gamma}\mathbf{u}(\gamma). \end{aligned}$$

Proof: From (3.5), $\Delta\mathbf{u}(\gamma) = \mathbf{u}(\gamma) - \mathbf{u}(0) = \mathbf{K}\mathbf{p}(\gamma)$. Since $\mathbf{K}$ is constant, differentiating with respect to γ yields the first equation of the theorem. Since $\mathbf{K}$ is also nonsingular, this equation may be inverted, yielding the second equation. □

3.2.2 Sliding mode

Bodies 1 and 2 are sliding relative to one another when the tangential component of relative contact velocity $\mathbf{u}(\gamma)$ is nonzero. In this case it is possible to derive a first order differential equation for $\mathbf{u}$, using p_z as the collision parameter.

Definition 2 *Let θ be the relative sliding direction at some point during a collision, that is, $\theta = Tan^{-1}(u_y, u_x)$, where the arctangent returns values in four quadrants. Define the* **sliding vector** $\boldsymbol{\xi}(\theta)$ *as*

$$\boldsymbol{\xi}(\theta) = \begin{bmatrix} -\mu\cos\theta \\ -\mu\sin\theta \\ 1 \end{bmatrix} = \begin{bmatrix} -\mu u_x/\sqrt{u_x^2+u_y^2} \\ -\mu u_y/\sqrt{u_x^2+u_y^2} \\ 1 \end{bmatrix}$$

where μ is the coefficient of friction.

The significance of the sliding vector is seen in the next lemma.

Lemma 2 *Let $\theta(p_z)$ be the relative sliding direction during a collision. While the bodies are sliding relative to each other,*

$$\frac{d}{dp_z}\mathbf{p}(p_z) = \boldsymbol{\xi}(\theta(p_z)).$$

Proof: By the chain rule,

$$\frac{d}{dp_z}\mathbf{p}(p_z) = \frac{d}{dt}\mathbf{p}(p_z)\ \frac{dt}{dp_z}(p_z) = \mathbf{f}(p_z)\ \frac{dt}{dp_z}(p_z),$$

where $\mathbf{f}(p_z)$ is the collision force, the time derivative of impulse. By the Coulomb friction law,

$$\mathbf{f}(p_z) = \begin{bmatrix} -\mu\cos\theta(p_z) f_z(p_z) \\ -\mu\sin\theta(p_z) f_z(p_z) \\ f_z(p_z) \end{bmatrix}$$

under sliding conditions. Combining these equations and noting that $(dt/dp_z)(p_z) = 1/f_z(p_z)$ proves the lemma. □

Combining Theorem 7 and Lemma 2 produces:[1]

Theorem 8 (sliding ODE w.r.t p_z) *While two colliding bodies are in sliding contact, the derivatives of* $\mathbf{u}$ *with respect to* p_z *are given by*

$$\frac{d}{dp_z}\mathbf{u} = \mathbf{K}\ \boldsymbol{\xi}(\theta) = \mathbf{K}\begin{bmatrix} -\mu u_x/\sqrt{u_x^2+u_y^2} \\ -\mu u_y/\sqrt{u_x^2+u_y^2} \\ 1 \end{bmatrix}$$

This system of three nonlinear, first-order ODEs describe how the relative velocity evolves during a collision. By numerically integrating these equations using p_z as the independent variable, $\mathbf{u}$ can be tracked over the course of the collision, as long as the sliding velocity is nonzero. Figure 3.3 shows solution trajectories of the ODE system projected into the u_x-u_y plane. For this example, μ was chosen as 0.2, and

$$\mathbf{K} = \begin{bmatrix} +8 & -2 & +1 \\ -2 & +3 & -1 \\ +1 & -1 & +5 \end{bmatrix}$$

The diamonds in the figure mark different initial values for u_x and u_y at the beginning of the collision. The flow lines show how the relative tangential velocity evolves during the collision. For a given collision, only one of the flow lines is followed.

One counterintuitive aspect of Figure 3.3 is that many of the flow lines diverge from the origin, indicating that the magnitude of the sliding velocity increases during collision.

[1] The explicit dependence of $\mathbf{u}$ and θ on p_z is eliminated for clarity.

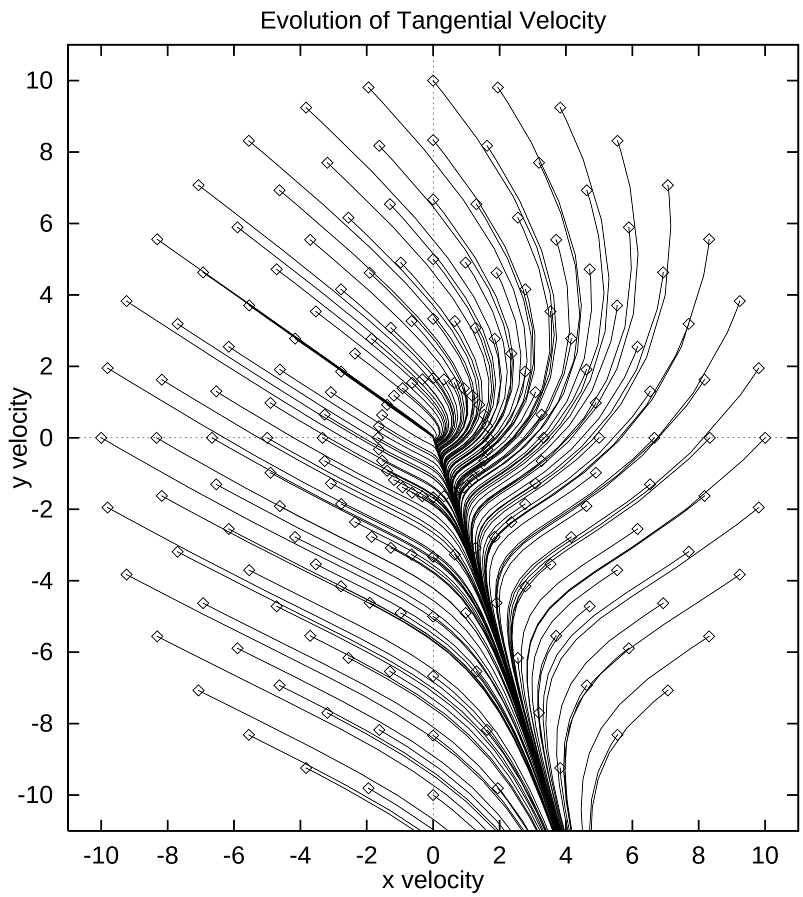

Figure 3.3: *Solution trajectories of the ODE system of Theorem 8 projected into the* u_x-u_y *plane.*

This behavior seems contrary to the laws of friction, which stipulate frictional force oppose sliding. The behavior is understood from Figure 3.4. In this case, the tangential velocity is zero at the beginning of the collision, but nonzero at the end of the collision. Although the frictional force opposes the motion, it can not overcome the effects of the normal force, which induces an angular velocity that causes the rod to slip.

3.2.3 Sticking mode

When the tangential component of relative contact velocity $\mathbf{u}$ vanishes, the bodies are sticking, and the analysis of the previous section 3.2.2 is not valid. Under sticking conditions, the Coulomb friction model does not completely determine the frictional force, but only requires that its magnitude not exceed μ times the magnitude of the normal force.

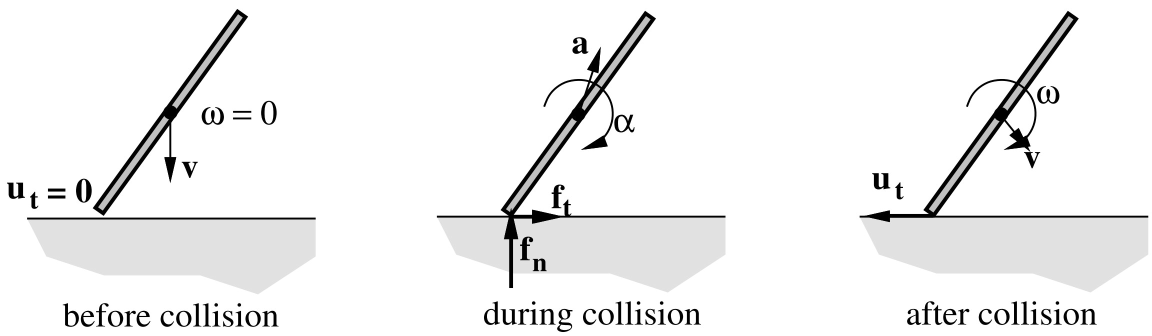

Figure 3.4: *A situation where the tangential relative contact velocity of the rod ($\mathbf{u}_t$) starts at zero and increases during the course of the collision, even though the frictional force resists this change in velocity.*

When sticking occurs, we assume, as does Routh, that if the frictional forces are strong enough to maintain the sticking condition, they will do so [Rou05]. If they are not strong enough, sliding will resume. The following theorem provides the means to test which of these behaviors occurs.

Theorem 9 (Stable sticking) *For a particular collision with collision matrix $\mathbf{K}$, the frictional collision forces can maintain sticking if and only if*

$$(K_{13}^{-1})^2 + (K_{23}^{-1})^2 \leq \mu^2 (K_{33}^{-1})^2,$$

where K_{ij}^{-1} denotes the (i, j) element of the inverse collision matrix $\mathbf{K}^{-1}$.

Proof: Consider the collision force $\mathbf{f}(p_z)$ at point p_z in the collision. From the Coulomb friction law, if there is no sliding the components of $\mathbf{f}$ must satisfy

$$f_x(p_z)^2 + f_y(p_z)^2 \leq \mu^2 f_z(p_z)^2,$$

or, expressing force as the time derivative of accumulated impulse,

$$\left[\frac{dp_x}{dt}(p_z)\right]^2 + \left[\frac{dp_y}{dt}(p_z)\right]^2 \leq \mu^2 \left[\frac{dp_z}{dt}(p_z)\right]^2.$$

Multiplying the above inequality by $(dt/dp_z)^2$:

$$\left[\frac{dp_x}{dp_z}(p_z)\right]^2 + \left[\frac{dp_y}{dp_z}(p_z)\right]^2 \leq \mu^2 \left[\frac{dp_z}{dp_z}(p_z)\right]^2.$$

If the sticking condition is stable, the derivatives of relative tangential velocity vanish: $u'_x(p_z) = u'_y(p_z) = 0$. In this case, Theorem 7 implies

$$\frac{d}{dp_z}\mathbf{p}(p_z) = \mathbf{K}^{-1}\begin{bmatrix} 0 \\ 0 \\ u'_z(p_z) \end{bmatrix} = \begin{bmatrix} K_{13}^{-1} \\ K_{23}^{-1} \\ K_{33}^{-1} \end{bmatrix} u'_z(p_z). \tag{3.6}$$

Combining the previous two equations proves the theorem. □

3.3 Collision integration

The collision response calculation is based on tracking the relative contact velocity $\mathbf{u}(\gamma)$ during a collision. Figure 3.5 depicts relative contact velocity space; each point (u_x, u_y, u_z) in this space corresponds to a particular relative contact point velocity between body 1 and body 2. The figure also depicts the trajectories $\mathbf{u}(\gamma)$ for three different collisions. For all of these collisions, $u_z < 0$ at the beginning of the trajectory since that the bodies are moving toward each other, and $u_z > 0$ at the end of the trajectory since the bodies are separating. The plane $u_z = 0$ is the plane of maximum compression; as $\mathbf{u}(\gamma)$ crosses this plane, the collision phase changes from compression to restitution (c.f. Figure 3.1). As long u_x and u_y are not both zero, $\mathbf{u}(\gamma)$ is evolved according to sliding equations, such as those given by Theorem 8. Path A is a case for which sliding occurs throughout the collision. If while tracking $\mathbf{u}(\gamma)$, u_x and u_y both vanish, then $\mathbf{u}(\gamma)$ lies on the u_z axis, also called the *line of sticking*. In this case, Theorem 8 is not applicable; instead the test of Theorem 9 is performed to determine if the sticking condition is stable. If it is, $\mathbf{u}(\gamma)$ will remain on the line of sticking for the duration of the collision; path C exhibits this behavior. If the sticking is not stable, meaning the frictional forces are not strong enough to prevent tangential slip, the point $\mathbf{u}(\gamma)$ leaves the line of sticking, again moving according to sliding equations. Path B exhibits this behavior. The question of exactly *how* $\mathbf{u}(\gamma)$ leaves the line of sticking has not been discussed; this is a deep question that will be treated in Section 3.4. If the paths of Figure 3.5 are projected into the u_x-u_y plane, one obtains flow lines like the ones in Figure 3.3.

This process of tracking the point $\mathbf{u}(\gamma)$ during collision is called *collision integration*. Given $\mathbf{u}(\gamma_0)$, the initial relative contact point velocity, and $\mathbf{u}(\gamma_f)$, the value of

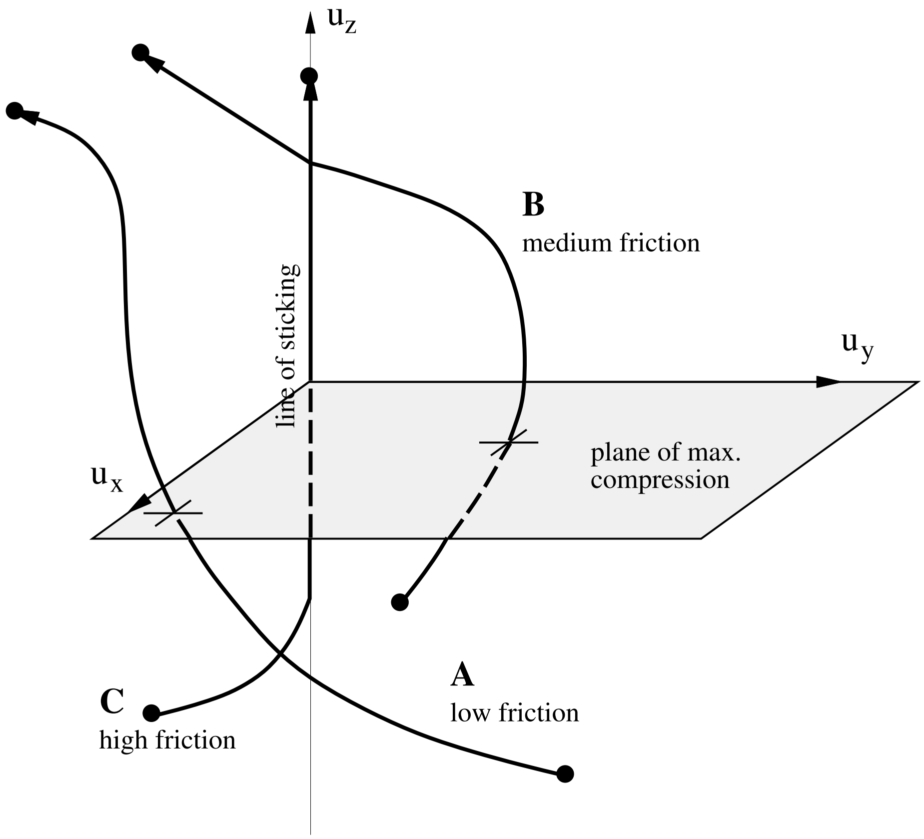

Figure 3.5: *Trajectories through relative contact velocity space for three different collisions.*

relative contact point velocity at the end of the collision, the collision impulse may be easily determined. Inverting (3.5),

$$\mathbf{p}(\gamma) = \mathbf{K}^{-1}\Delta\mathbf{u}(\gamma) = \mathbf{K}^{-1}\left(\mathbf{u}(\gamma_f) - \mathbf{u}(\gamma_0)\right),$$

Since $\mathbf{p}(\gamma_f)$ is the total impulse delivered during the collision, computing $\mathbf{p}(\gamma_f)$ is the ultimate goal of the collision resolution system. The collision integration process described above is somewhat simplified, and the purpose of this section is to describe the process in more detail. Procedures to handle stable and transient sticking are developed, as well as the termination conditions that indicate when the integration is to stop. The latter are based on Stronge's hypothesis for restitution.

3.3.1 Work done by collision forces

Consider a force $\mathbf{f}$ applied to a particle that moves along a continuous path $\mathbf{x}(t)$ through space. The total work performed by this force on the particle over the time interval $[0, t_f]$ is given by

$$W = \int_{t_0}^{t_f} \mathbf{f}(t) \cdot \dot{\mathbf{x}}(t)\ dt$$

If $\mathbf{x}(t)$ is a line segment and $\mathbf{f}(t)$ a constant force directed along this segment, this boils down to the familiar "work equals forces times distance" rule. During a collision, $\mathbf{f}$ is a collision force applied to a body at the contact point, and $\dot{\mathbf{x}}$ is the absolute velocity of the contact point, for example $\mathbf{u}_1$ on body 1. The formulation for work given above is not usable during a rigid body collision, because the collision forces are infinite and the time interval is infinitesimal. This can be remedied with the familiar tactic of re-parameterizing the collision with a different variable γ. With this change of variables,

$$W_1 = \int_0^{\gamma_f} \mathbf{f}(\gamma) \cdot \mathbf{u}_1(\gamma) \left(\frac{dt}{d\gamma}\right)\ d\gamma.$$

Replacing $\mathbf{f}(\gamma)$ with $(d/dt)\mathbf{p}(\gamma)$, applying the chain rule, and rearranging gives

$$W_1 = \int_0^{\gamma_f} \mathbf{u}_1(\gamma) \cdot \frac{d}{d\gamma}\mathbf{p}(\gamma)\ d\gamma = \int_0^{\gamma_f} \mathbf{u}_1(\gamma) \cdot \mathbf{p}'(\gamma)\ d\gamma.$$

This expresses the work done by the collision impulse on body 1. Similarly, the work done by the reaction impulse on body 2 is given by

$$W_2 = -\int_0^{\gamma_f} \mathbf{u}_2(\gamma) \cdot \mathbf{p}'(\gamma)\ d\gamma.$$

Combining the above two equations, and recalling that $\mathbf{u} = \mathbf{u}_1 - \mathbf{u}_2$, the total work done by the collision impulse on both bodies is

$$W = \int_0^{\gamma_f} \mathbf{u}(\gamma) \cdot \mathbf{p}'(\gamma)\ d\gamma.$$

The differential form of this equation will be useful:

Theorem 10 *The derivative of work done by the collision impulse with respect to the collision parameter γ is given by*

$$\frac{dW}{d\gamma}(\gamma) = \mathbf{u}(\gamma) \cdot \mathbf{p}'(\gamma) = \mathbf{u}(\gamma) \cdot \mathbf{K}^{-1}\mathbf{u}'(\gamma).$$

Since $\mathbf{K}^{-1}$ is a known constant, and the path $\mathbf{u}(\gamma)$ completely determines $\mathbf{u}'(\gamma)$, Theorem 10 implies that knowing the complete path $\mathbf{u}(\gamma)$ is enough to specify the total work done on the colliding bodies. In fact, knowing even less is sufficient:

Theorem 11 (path independence of W) *Suppose the relative contact velocity* $\mathbf{u}(\gamma)$ *proceeds from* $\mathbf{u}(\gamma_0)$ *to* $\mathbf{u}(\gamma_f)$ *during a collision, over some arbitrary path; let* $\Delta\mathbf{u}(\gamma_f) = \mathbf{u}(\gamma_f) - \mathbf{u}(\gamma_0)$*. The total work done by the collision forces is independent of the path taken, and is given by*

$$W = \frac{1}{2}\left(\mathbf{u}(\gamma_f) + \mathbf{u}(\gamma_0)\right)^T \; \mathbf{K}^{-1} \; \Delta\mathbf{u}(\gamma_f).$$

Proof: From Theorem 10,

$$\frac{dW}{d\gamma}(\gamma) = \mathbf{u}(\gamma)^T \; \mathbf{K}^{-1} \; \mathbf{u}'(\gamma) = \frac{1}{2}\left[\mathbf{u}(\gamma)^T \; \mathbf{K}^{-1} \; \mathbf{u}'(\gamma) + \mathbf{u}(\gamma)^T \; \mathbf{K}^{-1} \; \mathbf{u}'(\gamma)\right].$$

Transposing the second term in brackets—a scalar—and observing the symmetry of $\mathbf{K}^{-1}$,

$$\frac{dW}{d\gamma}(\gamma) = \frac{1}{2}\left[\mathbf{u}(\gamma)^T \; \mathbf{K}^{-1} \; \mathbf{u}'(\gamma) + \mathbf{u}'(\gamma)^T \; \mathbf{K}^{-1} \; \mathbf{u}(\gamma)\right].$$

Since $\mathbf{K}^{-1}$ is a constant,

$$\frac{dW}{d\gamma}(\gamma) = \frac{1}{2}\frac{d}{d\gamma}\left[\mathbf{u}(\gamma)^T \; \mathbf{K}^{-1} \; \mathbf{u}(\gamma)\right].$$

By the Fundamental Theorem of Calculus, the total work done over the interval $[0, \gamma_f]$ is $\left[\mathbf{u}(\gamma)^T\mathbf{K}^{-1}\mathbf{u}(\gamma)\right]|_0^{\gamma_f}$. Evaluating this expression,

$$\begin{aligned}
W &= \mathbf{u}(\gamma_f)^T\mathbf{K}^{-1}\mathbf{u}(\gamma_f) - \mathbf{u}(\gamma_0)^T\mathbf{K}^{-1}\mathbf{u}(\gamma_0) \\
&= \mathbf{u}(\gamma_f)^T\mathbf{K}^{-1}\mathbf{u}(\gamma_f) - \mathbf{u}(\gamma_f)^T\mathbf{K}^{-1}\mathbf{u}(\gamma_0) + \mathbf{u}(\gamma_f)^T\mathbf{K}^{-1}\mathbf{u}(\gamma_0) - \mathbf{u}(\gamma_0)^T\mathbf{K}^{-1}\mathbf{u}(\gamma_0) \\
&= \mathbf{u}(\gamma_f)^T\mathbf{K}^{-1}\Delta\mathbf{u} + \Delta\mathbf{u}^T\mathbf{K}^{-1}\mathbf{u}(\gamma_0).
\end{aligned}$$

Finally, transposing the second scalar term of the final equation yields the desired result.
□

An important corollary concerns a special energy preserving collision.

Corollary 1 *If during a collision, the relative contact velocity is exactly reversed, that is* $\mathbf{u}(\gamma_f) = -\mathbf{u}(\gamma_0)$*, then the collision forces do no net work on the colliding bodies, and the energy of the system is therefore unchanged by the collision.*

Stronge's hypothesis is used to determine when the collision integration should stop. This hypothesis is not a statement about the total work done by collision forces, but about the *normal work*, that is, the work done by the normal component of the collision force. The normal work is denoted W_z.

Theorem 12 *The derivative of the normal work with respect to the collision parameter γ is given by*

$$\frac{dW_z}{d\gamma}(\gamma) = u_z(\gamma)\ p_z'(\gamma).$$

Proof: From Theorem 10 and the chain rule,

$$\frac{dW}{d\gamma}(\gamma) = \mathbf{u}(\gamma) \cdot \mathbf{p}'(\gamma) = \mathbf{u}(\gamma) \cdot \mathbf{f}(\gamma)\frac{dt}{d\gamma}.$$

The differential work done only by the normal (z-component) of the force is therefore

$$\frac{dW}{d\gamma}(\gamma) = u_z(\gamma) f_z(\gamma)\frac{dt}{d\gamma} = u_z(\gamma) p_z'(\gamma).$$

□

3.3.2 Integrating collisions using different parameters

To this point, the only quantity mentioned as a possible collision parameter is p_z, the normal component of impulse. This is a natural choice since p_z is clearly monotonically increasing during the collision. There are, however, other choices for the collision parameter γ, and most of the results derived thus far apply equally well for other valid parameterizations. For a parameter γ to be valid, it must be monotonically increasing during the collision, and the quantities being integrated, such as $\mathbf{u}(\gamma)$ must be continuous. These properties have already been shown for the parameter p_z, and by the chain rule implies that any other parameter γ with the property that $d\gamma/dp_z$ is positive and finite is also valid. As an example, consider the parameter u_z, the normal component of relative velocity. From Theorem 8,

$$\frac{d}{dp_z}u_z = [K_{31}, K_{32}, K_{33}]\,\boldsymbol{\xi}(\theta).$$

As long as the right hand side is positive, u_z is a valid parameter.[2] As a second example, consider the parameter W_z, the normal work done during the collision. Applying Theo-

[2]This is true for almost all collisions, independent of the sliding angle θ. Section 3.4.2 deals with the case when it is not.

rem 12 with $\gamma = p_z$,

$$\frac{d}{dp_z} W_z = u_z \frac{d}{dp_z} p_z = u_z. \tag{3.7}$$

Therefore W_z is a valid parameter as long as $u_z > 0$, this is the case, for instance, throughout the restitution phase of the collision.

Since the equations of collision have already been developed using p_z as a parameter, why bother with other parameterizations? The reason has to do with how collision integrations are terminated. Stronge's hypothesis (Assumption 2) dictates the process by which the collision integration should proceed:

1. Integrate the compression phase of the integration, which terminates at the point of maximum compression when $u_z = 0$. Record the normal work done during this phase.

2. Integrate the restitution phase of the integration, which terminates when the normal work done during restitution reaches $-e^2$ times the normal work done during compression.

Numerical integrators integrate over an interval $[a, b]$, and the endpoints must be specified when the integrator is called. The values of p_z at the end of the compression phase and at the end of the restitution phase are not known *a priori*; hence, p_z is not a useful parameter. On the other hand, the value of u_z is known at the end of the compression phase: it is zero. Thus, u_z is a suitable parameter for the compression integration. At the end of the compression phase, the normal work done during compression is known, and from this the total work which must be done over the entire collision can be computed. The normal work W_z is therefore a suitable parameter for the restitution integration. The next sections reformulates the equations of collision for these new parameters, and also show how sticking is handled. It will be useful to partition the collision matrix $\mathbf{K}$ into row vectors:

$$\mathbf{K} = \left[\begin{array}{c} \mathbf{k}_x \\ \hline \mathbf{k}_y \\ \hline \mathbf{k}_z \end{array} \right].$$

The expressions K_{ij} and K_{ij}^{-1} respectively denote the (i, j)th entries of $\mathbf{K}$ and $\mathbf{K}^{-1}$.

3.3.3 Sliding mode under u_z and W_z parameterizations

The equations of sliding must be reformulated for the new parameterizations. The compression integration is performed with u_z as a parameter. During this integration, u_x

and u_y are considered functions of u_z. Since the value of W_z at the end of compression is needed, this quantity must also be tracked as a function of u_z. Of course u_z need not be explicitly integrated, since it is the independent variable. From the chain rule and Theorem 8,

$$\frac{du_x}{du_z} = \frac{du_x/dp_z}{du_z/dp_z} = \frac{\mathbf{k}_x\boldsymbol{\xi}(\theta)}{\mathbf{k}_z\boldsymbol{\xi}(\theta)}.$$

The derivative du_y/du_z is derived analogously. Lastly, from Theorem 12 with $\gamma = u_z$,

$$\frac{dW_x}{du_z} = u_z\frac{dp_z}{du_z} = \frac{u_z}{\mathbf{k}_z\boldsymbol{\xi}(\theta)}.$$

Summarizing the results:

$$\frac{d}{du_z}\begin{bmatrix} u_x \\ u_y \\ W_z \end{bmatrix} = \frac{1}{\mathbf{k}_z\boldsymbol{\xi}(\theta)}\begin{bmatrix} \mathbf{k}_x\boldsymbol{\xi}(\theta) \\ \mathbf{k}_y\boldsymbol{\xi}(\theta) \\ u_z \end{bmatrix} \tag{3.8}$$

The restitution integration is performed with W_z as a parameter. The three components of $\mathbf{u}$ must be integrated with respect to this variable. From Theorem 7 and the chain rule,

$$\frac{d}{dW_z}\mathbf{u} = \mathbf{K}\frac{d}{dW_z}\mathbf{p} = \mathbf{K}\frac{d}{dp_z}\mathbf{p}\ \frac{dp_z}{dW_z}.$$

From (3.7), $dp_z/dW_z = u_z^{-1}$; as a result,

$$\frac{d}{dW_z}\mathbf{u} = \frac{1}{u_z}\mathbf{K}\boldsymbol{\xi}(\theta) = \frac{1}{u_z}\begin{bmatrix} \mathbf{k}_x\xi(\theta) \\ \mathbf{k}_y\xi(\theta) \\ \mathbf{k}_z\xi(\theta) \end{bmatrix}. \tag{3.9}$$

The above equation is valid for $u_z > 0$, but it is not well behaved near $u_z = 0$. To avoid using this equation near this singularity, u_z is retained as the integration variable slightly beyond the compression phase, into the early restitution phase. After u_z is safely positive, the restitution integration can complete using W_z as a parameter. Care must be taken that the small *extension* integration with respect to u_z does not cause W_z to exceed its final value. The following lemma computes a safe upper limit for u_z for the extension integration. The complete life cycle of a collision is depicted in Figure 3.6.

Lemma 3 *Let W_r denote the normal work which to be done during the entire restitution phase. Suppose integration with respect to u_z is extended into the restitution phase, until $u_z = b$, where*

$$b = \sqrt{2W_r\left(K_{33} - \mu\sqrt{K_{31}^2 + K_{32}^2}\right)}.$$

Then the normal work will not exceed W_r during this extension integration.

Proof: Suppose $dW_z/du_z \leq Cu_z$ for some positive constant C. Separating the variables and integrating as u_z varies over $[0, b]$ yields

$$\Delta W_z \leq \frac{1}{2}Cb^2.$$

To insure that ΔW_z does not exceed W_r, it suffices to choose

$$b = \sqrt{2W_r/C}.$$

All that is needed is C. From the last of Equations (3.8),

$$\frac{dW_z}{du_z} = \frac{u_z}{\mathbf{k}_z \boldsymbol{\xi}(\theta)} \leq \left(K_{33} - \mu\sqrt{K_{31}^2 + K_{32}^2}\right)^{-1} u_z.$$

The coefficient of u_z on the right hand side is a valid choice for C; the lemma follows. □

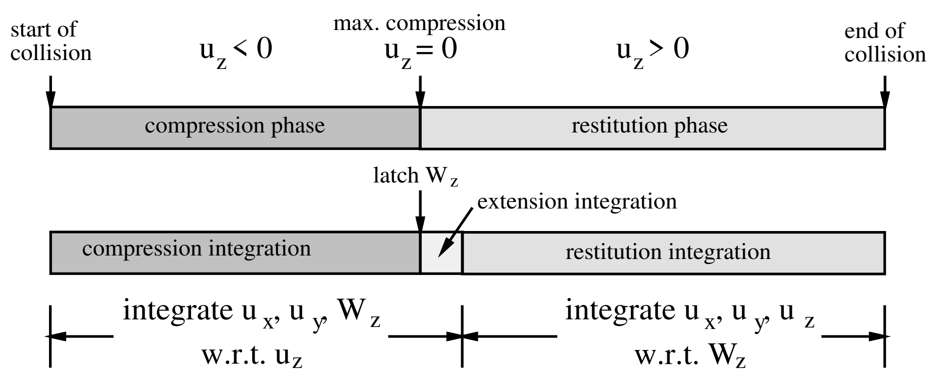

Figure 3.6: *The physical life cycle of a collision, and the corresponding collision integration. The compression and extension integrations are performed using (3.8), and the restitution integration is performed using (3.9). The value of W_z latched at the point of maximum compression is used to determine the upper limit of W_z that ends the restitution integration, according to Stronge's hypothesis. The value of u_z that terminates the extension integration is given by Lemma 3.*

3.3.4 Handling sticking during collision integration

We have not yet discussed what happens if sticking occurs at some point during the compression, extension, or restitution integrations. Practically, sticking is detected when the magnitude of the tangential component of relative contact velocity falls below some tolerance ε_s. At this point, the collision resolution algorithm determines if the sticking is stable or not (Theorem 9), and proceeds accordingly. In either case, the normal sliding integration halts, and the evolution of the variables for the remainder of the collision may be computed by solving simple algebraic equations. The solutions of three first order ODEs will be useful; the claims of the following observation are verified by separating variables and integrating [BD86].

Observation 1 (solutions of first order ODEs) *Let y be a differentiable function of an independent variable γ that varies over the $[a, b]$, and C be a constant. Then:*

$$\frac{dy}{d\gamma} = C \quad \Rightarrow \quad y(b) = y(a) + C(b-a) \tag{3.10}$$

$$\frac{dy}{d\gamma} = C\gamma \quad \Rightarrow \quad y(b) = y(a) + \frac{1}{2}C\left(b^2 - a^2\right) \tag{3.11}$$

$$\frac{dy}{d\gamma} = \frac{C}{y} \quad \Rightarrow \quad y(b) = \sqrt{y(a)^2 + 2C(b-a)} \tag{3.12}$$

First consider the case of stable sticking. There are two varieties, depending on whether the sticking was detected during integration with respect to u_z or W_z integration. Suppose a u_z integration was in progress. Let a be the current value of u_z, when sticking was detected, and let b be the upper limit of u_z for the current integration ($b = 0$ if sticking was detected during the compression integration, or a positive number if sticking was detected during the extension integration). The goal is to determine $u_x(b)$, $u_y(b)$, and $W_z(b)$. Stable sticking implies $u_x(b) = u_y(b) = 0$. By Theorem 12,

$$\frac{dW_z}{du_z} = u_z \, \frac{dp_z}{du_z}.$$

Since $d\mathbf{u}/du_z = (0, 0, 1)^T$ under stable sticking, Theorem 7 implies $dp_z/du_z = K_{33}^{-1}$. Thus

$$\frac{dW_z}{du_z} = K_{33}^{-1} u_z, \tag{3.13}$$

and applying the second rule of Observation 1 gives

$$W_z(b) = W_z(a) + \frac{1}{2}K_{33}^{-1}\left(b^2 - a^2\right).$$

Next suppose that stable sticking prevails at the beginning of the restitution integration, or that it occurs during it. Let a be the current value of W_z and b be the final value. The goal is to compute $\mathbf{u}(b)$, and again $u_x(b) = u_y(b) = 0$ because sticking is stable. Rewriting (3.13) as

$$\frac{du_z}{dW_z} = \frac{1/K_{33}^{-1}}{u_z},$$

and applying the third rule of Observation 1 gives

$$u_z(b) = \sqrt{u_z(a)^2 + \frac{2}{K_{33}^{-1}}(b-a)}.$$

Instable sticking occurs if the contact force required to maintain the sticking lies outside the friction cone; in this case sliding must resume. How this happens is not obvious, but the trajectory plots in Figure 3.3 give a clue. Sticking occurs for this collision if the tangential velocity point reaches the origin. For the $\mathbf{K}$ and μ that generate that plot, sticking is instable and the tangential velocity point must leave the origin. In the plot, there are two rays along which the sliding direction is constant. Referenced to the positive x-axis, there is one ray at $147°$ and one ray at $-71°$. If the tangential velocity point is on either of these rays, the direction of sliding remains constant and the point never leaves the ray. The $147°$ ray is called a *converging ray*, since velocity point moves along it toward the origin, and the $-71°$ ray is a *diverging ray*, since the velocity point moves along it away from the origin. If the velocity point is not on one of these rays, the direction of sliding is continuously changing. The tangential velocity point can only leave the origin along a diverging ray. Leaving in any other direction would mean crossing flow lines or flowing in the wrong direction along a converging ray. For any $\mathbf{K}$ and μ, the sliding plot resembles the one in Figure 3.3, although the number of converging and diverging rays may vary. After instable sticking, sliding can only resume along a diverging ray, which implies that the direction of sliding is constant after instable sticking. Fortunately, resuming sliding is a deterministic process: *When sticking is instable, there is exactly one diverging ray along which sliding may resume.* The next section proves this claim, and gives an algorithm for computing the direction of the ray. Taking the claim on faith for now, call the direction of the unique diverging ray β, which can take on any value modulo 2π. To resume sliding after instable sticking, one option is to set the values of u_x and u_y to lie on the diverging ray, very close to the origin:

$$\begin{aligned} u_x &= \varepsilon \cos\beta \\ u_y &= \varepsilon \sin\beta \end{aligned}$$

for some very small ε. Then the normal sliding mode integration could be resumed as the tangential velocity flows away from the origin. A better approach is to use the knowledge that the sliding direction is constant after instable sticking, in order to solve for the subsequent motion of the velocity point algebraically.

First consider the case of instable sticking occurring during integration with respect to u_z. Let a be the current value of u_z, and b the upper limit of the integration. The goal is to compute $u_x(B), u_y(b)$, and $W_z(b)$. From the chain rule and Theorem 8,

$$\frac{du_x}{du_z} = \frac{du_x/dp_z}{du_z/dp_z} = \frac{\mathbf{k}_x\boldsymbol{\xi}(\theta)}{\mathbf{k}_z\boldsymbol{\xi}(\theta)}.$$

Since $\theta \equiv \beta$ after instable sticking, the right hand side is constant. Applying the first rule of Observation 1,

$$u_x(b) = u_x(a) + \frac{\mathbf{k}_x\boldsymbol{\xi}(\beta)}{\mathbf{k}_z\boldsymbol{\xi}(\beta)}(b-a).$$

The value $u_y(b)$ is derived analogously:

$$u_y(b) = u_y(a) + \frac{\mathbf{k}_y\boldsymbol{\xi}(\beta)}{\mathbf{k}_z\boldsymbol{\xi}(\beta)}(b-a).$$

From Theorems 12 and 8,

$$\frac{dW_z}{du_z} = u_z\frac{dp_z}{du_z} = \frac{u_z}{du_z/dp_z} = \frac{u_z}{\mathbf{k}_z\boldsymbol{\xi}(\beta)}. \tag{3.14}$$

Applying the second rule of Observation 1 gives

$$W_z(b) = W_z(a) + \frac{b^2-a^2}{2\mathbf{k}_z\boldsymbol{\xi}(\beta)}.$$

Finally, consider instable sticking occurring before or during the restitution integration. Let a be the current value of W_z, and b be the upper limit of the integration. The goal is to compute $\mathbf{u}(b)$. Inverting (3.14),

$$\frac{du_z}{dW_z} = \frac{\mathbf{k}_z\boldsymbol{\xi}(\beta)}{u_z}.$$

Applying the third rule of Observation 1 gives

$$u_z(b) = \sqrt{u_z(a)^2 + 2\mathbf{k}_z\boldsymbol{\xi}(\beta)(b-a)}.$$

Once $u_z(b)$ is known, $u_x(b)$ and $u_y(b)$ can again be computed using the same technique as before:

$$\begin{aligned} u_x(b) &= u_x(a) + \frac{\mathbf{k}_x\boldsymbol{\xi}(\beta)}{\mathbf{k}_z\boldsymbol{\xi}(\beta)}[u_z(b)-u_z(a)] \\ u_y(b) &= u_y(a) + \frac{\mathbf{k}_y\boldsymbol{\xi}(\beta)}{\mathbf{k}_z\boldsymbol{\xi}(\beta)}[u_z(b)-u_z(a)]. \end{aligned}$$

3.4 Sticking stability and rays of constant sliding

The previous section introduced the problem of deciding how to resume sliding when instable sticking is detected during a collision integration: the tangential velocity point leaves the origin along a diverging ray of constant sliding direction. For the plot of Figure 3.3 this is possible because there is exactly one such ray. Consider the tangential velocity flow in Figure 3.7. Here, there are no diverging rays, and so it is impossible for the

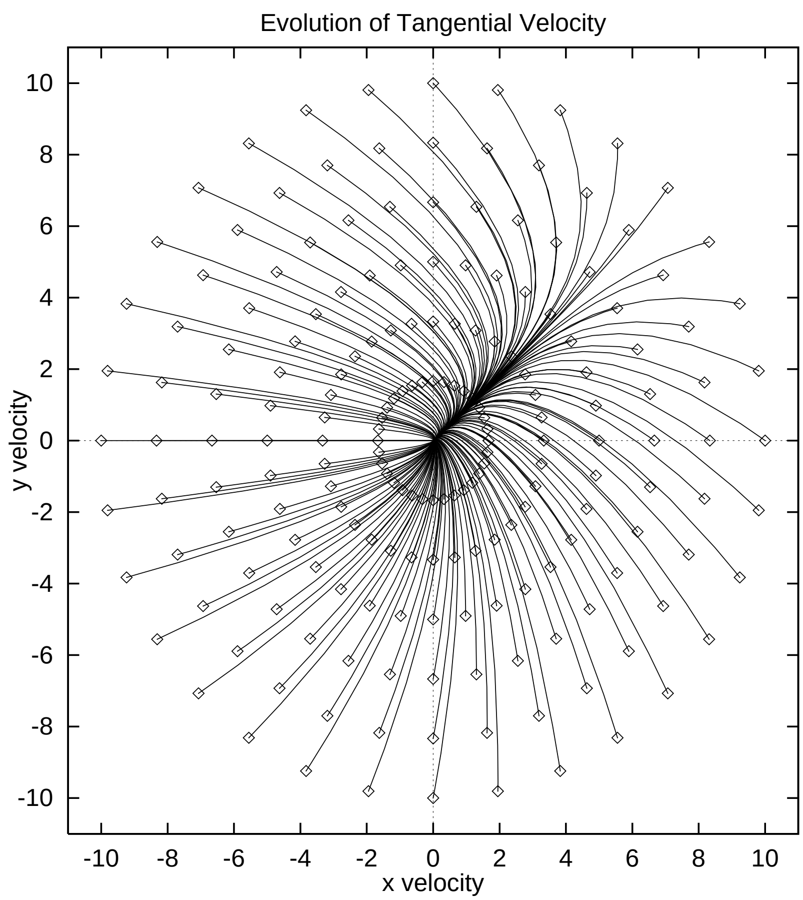

Figure 3.7: *A tangential velocity flow plot with two converging rays and no diverging rays.*

tangential velocity point to escape from the origin. For the **K** and μ used to generate this plot, sticking is stable, and so the lack of diverging rays is of no consequence. But can this situation arise in an instable sticking case? Next consider the velocity flow in Figure 3.8. Here, there would appear to be indeterminacy after instable sticking because there are two diverging rays. The **K** used to generate this plot is symmetric but not positive definite,

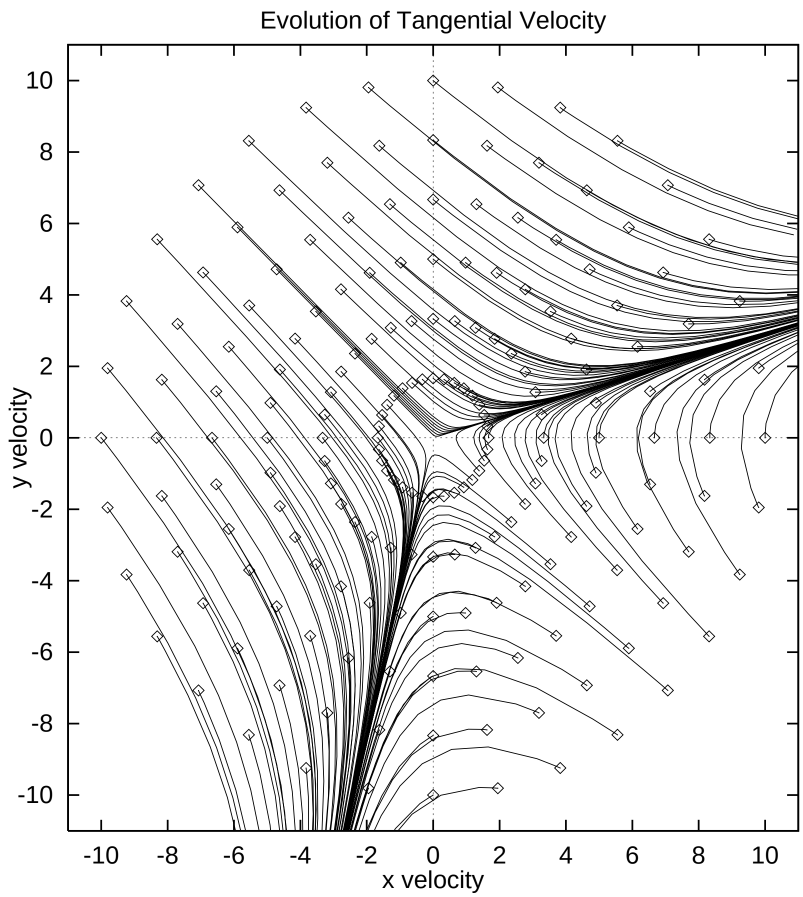

Figure 3.8: *A tangential velocity flow plot with two converging rays and two diverging rays.*

hence it is not even a valid choice. But it is important to verify that such indeterminacy can never occur with valid **K**. This section addresses these questions.

The issues addressed in this section have also been extensively studied by Bhatt and Koechling [BK94, BK96b, BK96a]. Their approach is to parameterize the space of all three-dimensional rigid body collisions into regions that are qualitatively similar, such as the set of collisions where there are no diverging rays. From their partitioning, the qualitative character of the collision can be determined *a priori* from the initial conditions. One of the main results of this section is a proof that the behavior is determinate in cases of instable sticking (Theorem 16). Bhatt and Koechling have also recently proved this long standing conjecture, using a purely algebraic method; the proof in this section is more geometric. Wang and Mason also partitioning collision space based on qualitative behavior, but for the case of two-dimensional collisions [WM87].

3.4.1 The $\mathbf{u}'$ ellipse

For the remainder of this section, the entries of the collision matrix are given by

$$\mathbf{K} = \begin{bmatrix} a & f & e \\ f & b & d \\ e & d & c \end{bmatrix}. \tag{3.15}$$

To study the evolution of sliding velocity during collision, the most convenient parameterization to use is the one based on p_z, and specified by Theorem 8. Using the above definition of $\mathbf{K}$, the theorem implies

$$\begin{bmatrix} u'_x \\ u'_y \end{bmatrix} = \mu \underbrace{\begin{bmatrix} a & f \\ f & b \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} -\cos\theta \\ -\sin\theta \end{bmatrix} + \underbrace{\begin{bmatrix} e \\ d \end{bmatrix}}_{\mathbf{s}}, \tag{3.16}$$

where $\mathbf{M}$ and $\mathbf{s}$ are the designated sub-matrices of $\mathbf{K}$. To simplify notation, the explicit dependence of $\mathbf{u}$ and θ on p_z is dropped in the rest of the analysis; primes denote derivatives with respect to p_z. Since only the tangential velocity is being studied, the the third (normal) component of relative velocity is dropped, so that $\mathbf{u} = (u_x, u_y)^T$, and $\mathbf{u}' = (u'_x, u'_y)^T$. Since $\mathbf{M}$ is a major sub-matrix of the positive definite matrix $\mathbf{K}$, it is also positive definite. Therefore,

$$\mathbf{M} = \mathbf{R}_\alpha \mathbf{M}_d \mathbf{R}_\alpha^T, \tag{3.17}$$

where $\mathbf{M}_d$ is a 2×2 diagonal matrix with positive entries along the diagonal, and $\mathbf{R}_\alpha$ is the orthogonal two dimensional rotation matrix for some angle α:

$$\mathbf{R}_\alpha = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}.$$

(See [HJ91] for justification of these assertions concerning positive definite matrices.) Combining results,

$$\begin{bmatrix} u'_x \\ u'_y \end{bmatrix} = \mu \mathbf{R}_\alpha \mathbf{M}_d \mathbf{R}_\alpha^T \begin{bmatrix} -\cos\theta \\ -\sin\theta \end{bmatrix} + \mathbf{s}.$$

As the sliding direction θ varies over $[0, 2\pi)$, $\mathbf{u}'$ traces out an ellipse in a counterclockwise direction. It is not the case that θ or $\mathbf{u}'$ actually vary in this manner during a collision, but the analysis to come is based on how the point $\mathbf{u}'$ behaves as a function of θ. Calling the diagonal entries of $\mathbf{M}_d$ a_d and b_d, the series of transformations indicated by the above equation is shown in Figure 3.9.

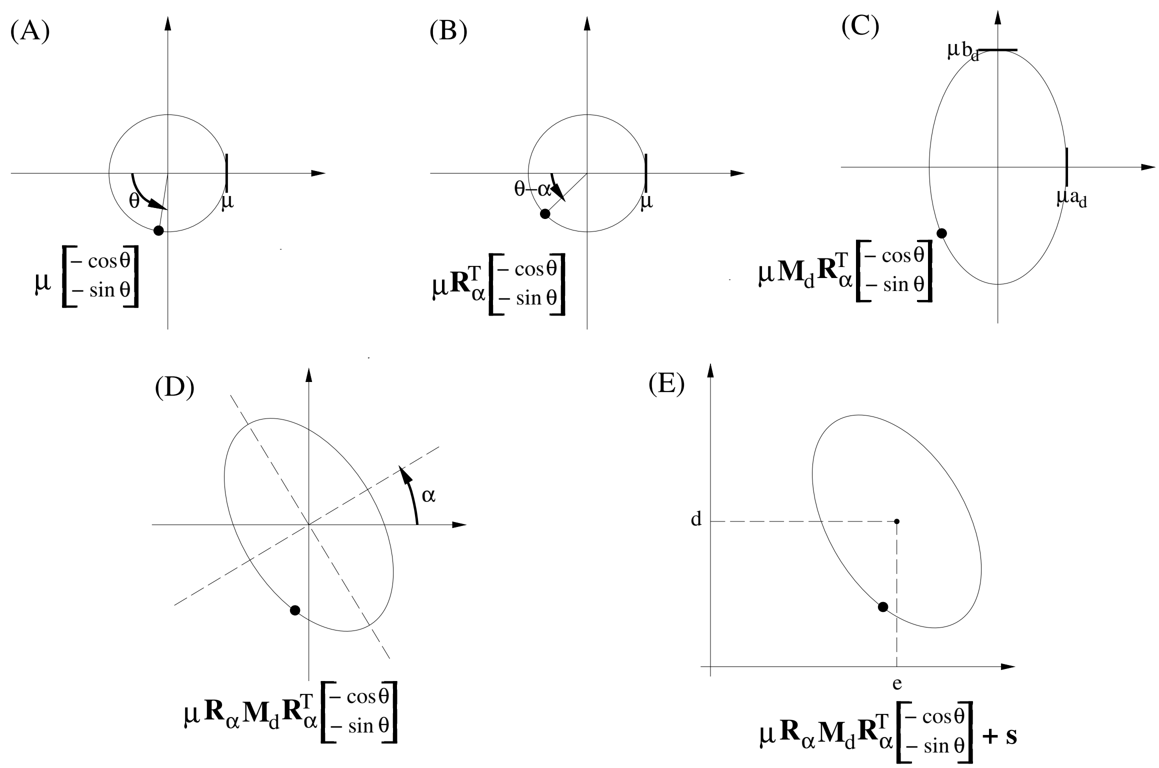

Figure 3.9: *A sequence of transformations applied to the point* $(-\cos\theta, -\sin\theta)^T$ *in order to produce the velocity derivatives* $(u'_x, u'_y)^T$*. The latter trace out a counter-clockwise ellipse as* θ *varies over* $[0, 2\pi)$.

Lemma 4 *The* $\mathbf{u}'$ *ellipse [shown in Figure 3.9(E)] circles the origin if and only if the entries of the collision matrix* $\mathbf{K}$ *satisfy*

$$(b^2 + f^2)e^2 + (a^2 + f^2)d^2 - 2(a+b)fde < \mu^2|\mathbf{M}|^2.$$

Proof: Establish a new frame $\mathcal{G}$ located at the center of the $\mathbf{u}'$ ellipse and rotated an angle α relative to the original frame $\mathcal{F}$, as shown in Figure 3.10. The coordinates of the origin (of frame $\mathcal{F}$) expressed in frame $\mathcal{G}$ are given by

$$\mathbf{o} = \begin{bmatrix} o_x \\ o_y \end{bmatrix} = \mathbf{R}(-\alpha)(-\mathbf{s}) = -\mathbf{R}_\alpha^T \mathbf{s}.$$

The ellipse encircles this point if and only if

$$\left(\frac{o_x}{a'}\right)^2 + \left(\frac{o_y}{b'}\right)^2 < \mu^2.$$

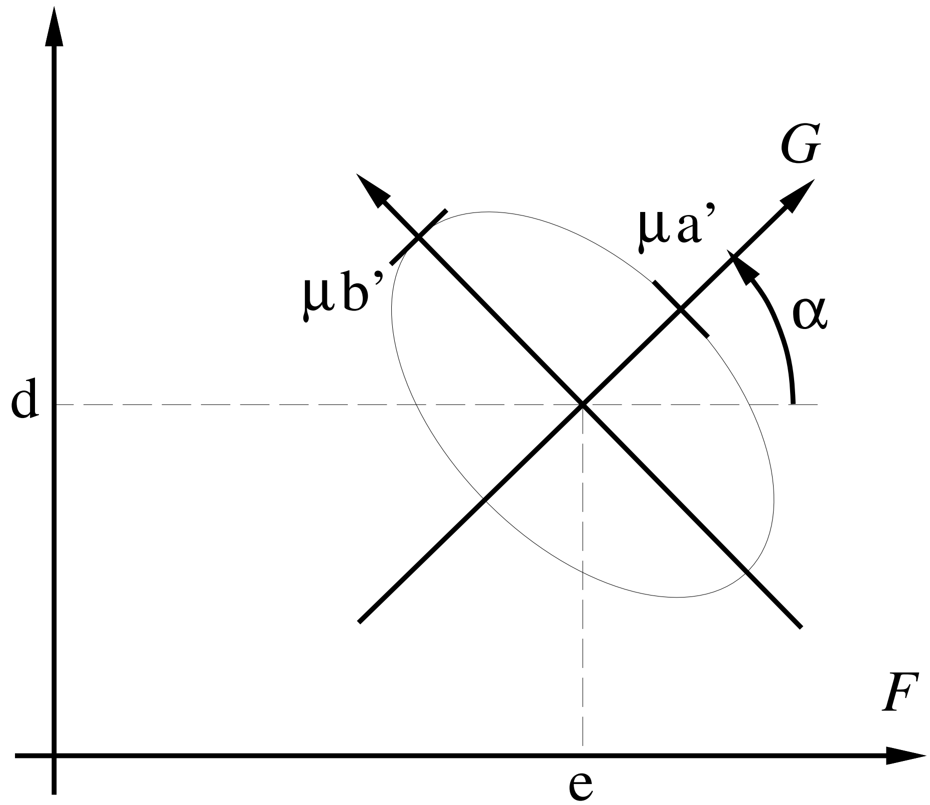

Figure 3.10: *Testing whether the* $\mathbf{u}'$ *ellipse circles the origin (Lemma 4).*

Since the diagonal elements of $\mathbf{M}_d^{-1}$ are $1/a'$ and $1/b'$, this criterion can be written

$$\|\mathbf{M}_d^{-1}\mathbf{o}\| = (\mathbf{M}_d^{-1}\mathbf{o})^T(\mathbf{M}_d^{-1}\mathbf{o}) = \mathbf{o}^T\mathbf{M}_d^{-2}\mathbf{o} < \mu^2.$$

From (3.17), $\mathbf{M}_d^{-1} = \mathbf{R}_\alpha^T\mathbf{M}^{-1}\mathbf{R}_\alpha$. Again rephrasing the criterion,

$$\left(-\mathbf{R}_\alpha^T\mathbf{s}\right)^T\left(\mathbf{R}_\alpha^T\mathbf{M}^{-1}\mathbf{R}_\alpha\right)^2\left(-\mathbf{R}_\alpha^T\mathbf{s}\right) < \mu^2,$$

which conveniently reduces to

$$\mathbf{s}^T\ \mathbf{M}^{-2}\ \mathbf{s} < \mu^2.$$

$\mathbf{M}^{-1}$ is given by

$$\mathbf{M}^{-1} = \frac{1}{|\mathbf{M}|}\begin{bmatrix} b & -f \\ -f & a \end{bmatrix},$$

and so the criterion becomes

$$\mathbf{s}^T\begin{bmatrix} b & -f \\ -f & a \end{bmatrix}^2\mathbf{s} < \mu^2|\mathbf{M}|^2.$$

Straightforward computation with $\mathbf{s} = (e, d)^T$ gives the lemma. □

The criterion of this lemma seems cryptic, but it can be expressed much more simply:

Theorem 13 *The* $\mathbf{u}'$ *ellipse circles the origin if and only if sticking is stable.*

Proof: By the direct inversion formula,

$$\mathbf{K}^{-1} = \frac{1}{|\mathbf{K}|}\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \begin{vmatrix} f & e \\ b & d \end{vmatrix} & -\begin{vmatrix} a & e \\ f & d \end{vmatrix} & \begin{vmatrix} a & f \\ f & b \end{vmatrix} \end{bmatrix},$$

where the dots mark entries which are irrelevant. Notice that the determinant in the lower right corner of the matrix is $|\mathbf{M}|$. In terms of the entries of $\mathbf{K}$, the stable sticking criterion given by Theorem 9 can be written

$$\left(\frac{\begin{vmatrix} f & e \\ b & d \end{vmatrix}}{|\mathbf{K}|}\right)^2 + \left(\frac{-\begin{vmatrix} a & e \\ f & d \end{vmatrix}}{|\mathbf{K}|}\right)^2 < \mu^2 \left(\frac{|\mathbf{M}|}{|\mathbf{K}|}\right)^2.$$

Multiplying through by $|\mathbf{K}|^2$ and expanding the determinants on the left hand side leads to the same criterion given in Lemma 4. □

3.4.2 Directions of constant sliding

Let $\mathrm{Tan}^{-1}(y, x)$ denote the four quadrant arctangent function which returns the unique angle ϕ in $[0, 2\pi)$ such that $y = r\sin\phi$ and $x = r\cos\phi$ for some positive constant r. The sliding direction θ is given by

$$\theta = \mathrm{Tan}^{-1}(u_y, u_x).$$

If $\mathbf{u}'$ is in the same direction as $\mathbf{u}$ or exactly opposed to it. Then one of the following holds:

$$\begin{aligned} \mathrm{Tan}^{-1}\left(u'_y, u'_x\right) &= \theta \\ \mathrm{Tan}^{-1}\left(u'_y, u'_x\right) &= \theta + \pi \pmod{2\pi}. \end{aligned}$$

If either of these relations hold, then the direction of $\mathbf{u}$ remains constant during collision integration. Values of θ which satisfy these relations define the location of diverging and converging rays, respectively. Let $\hat{\mathbf{u}} = (\cos\theta, \sin\theta)^T$ be the unit vector aligned with $\mathbf{u}$. Then one of the above relations is satisfied if and only if $\hat{\mathbf{u}} \times \mathbf{u}' = \mathbf{0}$. The particular relation which holds can be determined by the sign of $\hat{\mathbf{u}} \cdot \mathbf{u}'$: if it is positive, the first one holds; if it is negative, the second one holds. These results, along with (3.16) yield the following theorem.

Theorem 14 (location of rays) *Rays of constant sliding direction are located at angles θ satisfying the equation*

$$(\hat{\mathbf{u}} \times \mathbf{u}')(\theta) = (a-b)\mu\cos\theta\sin\theta + f\mu(\sin^2\theta - \cos^2\theta) + d\cos\theta - e\sin\theta = 0.$$

At such values of θ, if

$$(\hat{\mathbf{u}} \cdot \mathbf{u}')(\theta) = -a\mu\cos^2\theta - b\mu\sin^2\theta - 2f\mu\cos\theta\sin\theta + d\sin\theta + e\cos\theta$$

is positive, the ray is diverging, otherwise it is converging.

Corollary 2 *There can be no more than four rays of constant sliding direction.*

Proof: The cross product criterion of Theorem 14 can be converted to a polynomial equation by making a standard change of variables: $t = \tan^{-1}(\theta/2)$, $\cos\theta = (1-t^2)/(1+t^2)$, $\sin\theta = (2t)/(1+t^2)$. This leads to a fourth order polynomial equation in t, having at most four roots. $\square$

The above corollary also gives an efficient algorithm for computing the directions of the converging and diverging rays. The roots of a fourth order polynomial are computed algebraically. These give the directions of the rays, which are then easily classified as converging or diverging, based on the test of Theorem 14. The lighthouse analogy of Figure 3.11 is useful in understanding the rays of constant sliding direction. A lighthouse at the origin rotates counter-clockwise, sweeping a white beam across the plane. It also emits a red beam in the opposite direction of the white beam. The location white beam represents the sliding direction θ. The $\mathbf{u}'$ ellipsoid is also located in the plane, and as the θ varies, the point $\mathbf{u}'(\theta)$ moves around this ellipsoid in a counter-clockwise direction. Let $\beta(\theta)$ be the angle this point makes with the positive x-axis. If at a particular value θ_1 the

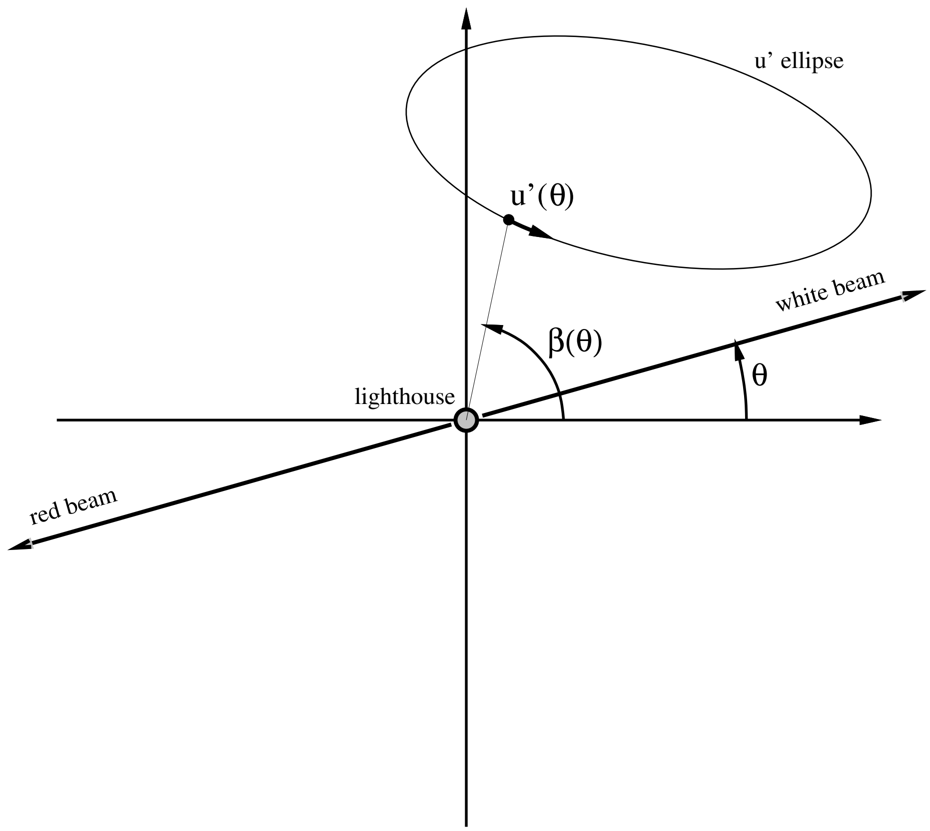

Figure 3.11: *The lighthouse analogy for rays of constant sliding direction.*

white beam directly illuminates the point $\mathbf{u}'$, there is a diverging ray at θ'. If at a particular value θ_2 the red beam directly illuminates the point $\mathbf{u}'$, then there is a converging ray θ_2.

Lemma 5 *Consider the point* $\mathbf{u}'$ *moving counter-clockwise around the ellipse shown in Figure 3.11 as* θ *goes from 0 to* 2π*. Then* $\beta(\theta) = \theta$ *implies* $d\beta/d\theta \leq 0$*. In other words: when the point is illuminated by the white beam,* $\beta(\theta)$ *can not be increasing.*

Proof: $\beta = \mathrm{Tan}^{-1}\left(u'_y, u'_x\right)$, and so

$$\frac{d\beta}{d\theta} = \frac{1}{1 + \left(u'_y/u'_x\right)^2} \frac{d}{d\theta}\left(\frac{u'_y}{u'_x}\right) = \frac{(du'_y/d\theta)u'_x - (du'_x/d\theta)u'_y}{{u'_x}^2 + {u'_y}^2}.$$

Therefore,

$$\mathrm{sgn}\left(\frac{d\beta}{d\theta}\right) = \mathrm{sgn}\left(\frac{du'_y}{d\theta}u'_x - \frac{du'_x}{d\theta}u'_y\right).$$

Differentiating (3.16) gives

$$\frac{d}{d\theta}\begin{bmatrix} u'_x \\ u'_y \end{bmatrix} = \begin{bmatrix} a\mu\sin\theta - f\mu\cos\theta \\ f\mu\sin\theta - b\mu\cos\theta \end{bmatrix},$$

and so

$$\operatorname{sgn}\left(\frac{d\beta}{d\theta}\right) = \operatorname{sgn}\left[(f\mu\sin\theta - b\mu\cos\theta)u'_x - (a\mu\sin\theta - f\mu\cos\theta)u'_y\right].$$

If $\beta(\theta) = \theta$ then $u'_x = r\cos\theta$ and $u'_y = r\sin\theta$ for some positive constant r. Substituting these values into the above equation, dividing the argument of the right hand sgn by the positive quantity μr yields:

$$\operatorname{sgn}\left(\frac{d\beta}{d\theta}\right) = \operatorname{sgn}(\underbrace{-b\cos^2\theta - a\sin^2\theta - 2f\cos\theta\sin\theta}_{\mathcal{D}}).$$

$\mathbf{M}$ is positive definite, so its determinant $ab - f^2$ is positive, and $\sqrt{ab} > |f|$. Thus,

$$\begin{aligned} \mathcal{D} &< -b\cos^2\theta - a\sin^2\theta + 2\sqrt{ab}\,|\cos\theta|\,|\sin\theta| \\ &= -(\sqrt{b}\,|\cos\theta| - \sqrt{a}\,|\sin\theta|)^2 \leq 0. \end{aligned}$$

This implies $d\beta/d\theta \leq 0$. $\square$

Theorem 15 *If sticking is stable, there are no diverging rays of constant sliding direction.*

Proof: If sticking is stable, the $\mathbf{u}'$ ellipse circles the origin (lighthouse) by Theorem 13. For this case, $d\beta/d\theta$ must be positive during the entire counter-clockwise circuit of the point $\mathbf{u}'$ around the ellipse, and therefore by Lemma 5, $\beta(\theta)$ can never equal θ. $\square$

Theorem 16 *If sticking is instable, there is exactly one diverging ray of constant sliding direction.*

Proof: If sticking is stable, the $\mathbf{u}'$ ellipse does not circle the origin (lighthouse) by Theorem 13. In this case, $\beta_{min} \leq \beta(\theta) \leq \beta_{max}$, with $\beta_{\max} - \beta_{\min} < \pi$. Clearly, as the white beam sweeps over the range $[\beta_{min}, \beta_{max}]$, it must illuminate the point $\mathbf{u}'$ at least once, and so there is at least one value of θ for which $\beta(\theta) = \theta$. To see that there is not more than one, consider the graph shown in Figure 3.12. Lemma 5 implies that the graph of $\beta(\theta)$ can

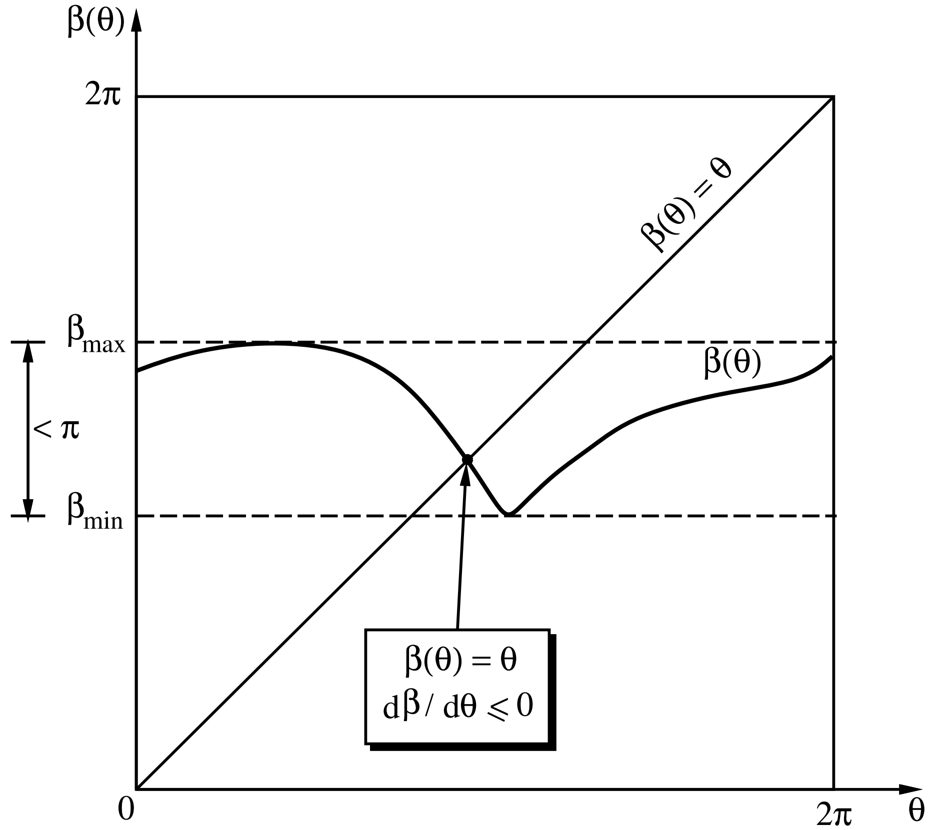

Figure 3.12: *Proving there can be no more than one diverging ray when sticking is instable.*

never cross the line $\beta(\theta) = \theta$ while $\beta(\theta)$ is increasing, and therefore it can cross only once. (More crossings would be possible if the graph of $\beta(\theta)$ "wrapped around" over an interval of 2π, but this is impossible since $\beta_{\rm max} - \beta_{\rm min} < \pi$.) The conclusion is that there is exactly one value of θ for which $\beta(\theta) = \theta$. $\Box$

Theorem 15 guarantees that Figure 3.7 is representative of all flow plot when sticking is stable: rays only converge to the origin; none diverge. Theorem 16 proves that when sticking is not stable, there is exactly one diverging ray, as in Figure 3.3. Thus, there is a well defined direction along which sliding resumes after instable sticking. This result had been previously conjectured in [BK94, MC95a]. Recently, Bhatt and Koechling have devised an alternative, algebraic proof of this result [BK96a]. The previous two theorems also prove that flows like the one of Figure 3.8 never occur for valid $\mathbf{K}$, which are positive definite.

A final loose end to tie up is how to handle situations where $du_z/d\gamma$ is negative at the beginning of a collision. This paradoxical situation occurs when the collision force initially accelerates the bodies *into* each other instead of away from each other. The phenomenon is well known; see [Bar89] for a specific example.

Theorem 17 *For values of θ for which the dot product discriminant of Theorem 14 is positive, du_z/dp_z is also positive.*

Proof: The hypothesis can be rephrased as

$$-a\mu^2\cos^2\theta - b\mu^2\sin^2\theta - 2f\mu^2\cos\theta\sin\theta + d\mu\sin\theta + e\mu\cos\theta > 0.$$

Since $\mathbf{K}$ is positive definite, $\boldsymbol{\xi}(\theta)^T\mathbf{K}\boldsymbol{\xi}(\theta)$ is positive. Expanding this product yields

$$a\mu^2\cos^2\theta + b\mu^2\sin^2\theta + 2f\mu^2\cos\theta\sin\theta + c - 2e\mu\cos\theta - 2d\mu\sin\theta > 0.$$

Adding these two inequalities,

$$-e\mu\cos\theta - d\mu\sin\theta + c > 0.$$

From Theorem 8, du_z/dp_z is positive exactly when the above inequality is satisfied. □

As the collision is integrated, the relative tangential velocity point asymptotically approaches the rays of constant sliding direction. This is evident in the flows of Figures 3.3, 3.7, and 3.8. If it approaches a diverging ray, then in some neighborhood of the ray the conditions of Theorem 17 will be met, and u_z will increase for the remainder of the collision. If it approaches a converging ray, the point will eventually reach the origin. If sticking is not stable, the point leaves along a diverging ray, and u_z will again increase for the duration of the collision. If sticking is stable, then by definition, $\mathbf{u}$ travels in the direction $(0,0,1)^T$ for the duration of the collision. Hence, in any case, u_z *eventually* increases, and remains increasing for the duration of the collision. To handle cases for which u_z is is initially decreasing—and these are extremely rare[3]—p_z is chosen as the initial integration parameter, and is used until du_z/dp_z becomes positive. At this point, the normal compression integration with respect to u_z can begin, and the rest of the collision integration is handled as before.

[3]In the vast majority of simulations performed with *Impulse*, including all of those described in the Chapter 7, collisions in which u_z is initially decreasing never occur.

3.5 Static contact and microcollisions

Under impulse-based simulation, continuous contact, such as that between a book and the table is rests on, is modeled through trains of collisions. In this case, the collision impulses model a constant reaction force which does no work on the stationary objects. The collision resolution method described thus far is not completely suitable for modeling such static contact forces. To understand the problem, consider the book on table example. Through collisions, the energy of the book steadily decreases, as the book sinks further and further into the collision envelope of the table. The decreased distance between the bodies causes time of impact estimates to decrease, increasing collision checks between the bodies. The simulator would be brought to a grinding halt as the separation distance and the time between impacts approached zero.

To alleviate this problem, *microcollisions* are proposed as a way of modeling static contact forces. Microcollision impulses are not computed in the standard way. For a collision to be a microcollision candidate, the initial relative normal velocity at the contact point must be small since static contact occurs only as objects settle onto one another; high relative normal velocities at the contact point correspond to colliding contact. In *Impulse*, this criterion is defined precisely as

$$-u_z(\gamma_0) < \sqrt{2g\varepsilon_c}, \tag{3.18}$$

where g is the acceleration of gravity. The right hand side is the velocity that an object acquires as it falls from rest through the collision envelope. Since the distance between objects in static contact is approximately ε_c, this seems a reasonable velocity bound.

For microcollisions, the coefficient of restitution is artificially increased based on the penetration of the bodies within the collision envelope, as shown in Figure 3.13. The graph shows e, the actual coefficient of restitution used in resolving the collision, as a function of the penetration depth δ, which is in the range $(0, \varepsilon_c)$. The value e_{def}, $0 \leq e_{\text{def}} \leq 1$, is the default value for e based on material properties of the objects. The value $e_{\max}$ is the maximum value e may assume. As shown in the graph, $e(\delta)$ is a second order curve, with vanishing derivative at $\delta = 0$. This is so that e can be made fairly large for deep penetrations, while still remaining close to its default value for most collisions, which have small penetrations. In *Impulse*, $e_{\max}$ was chosen as 5.0. This value is enough to prevent the simulation speed from slowing down noticeably when objects come to rest on other objects.

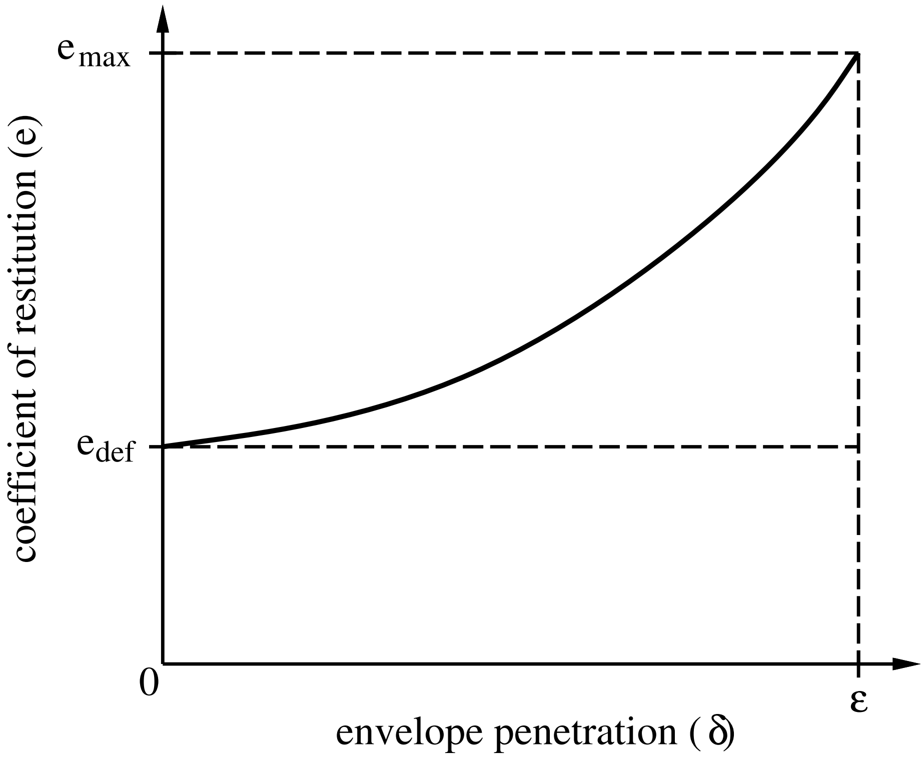

Figure 3.13: *The coefficient of restitution is artificially increased to prevent prolonged, deep penetration into the collision envelope.*

The resting object vibrates on its supporting object with a vibration amplitude on the order of ε_c; for most views, this amplitude is less than a pixel, and so the object appears at rest.

A related problem is that of a block on a ramp (Figure 3.14). If the coefficient

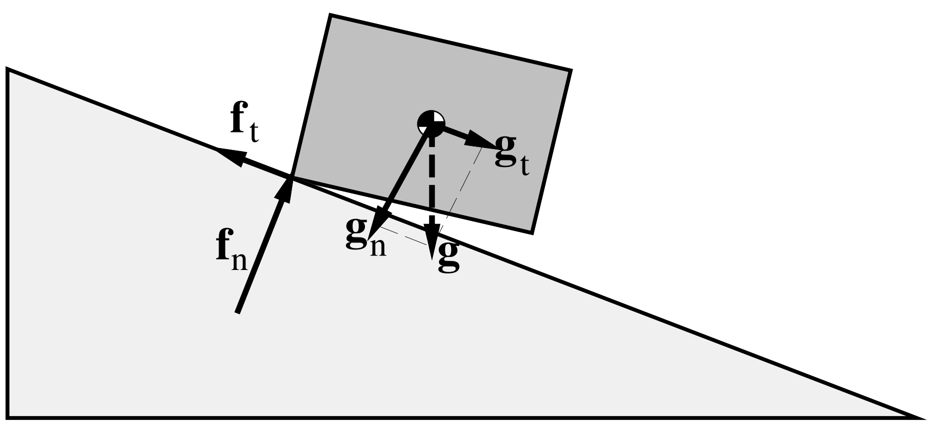

Figure 3.14: *Standard collision impulses will cause the block to creep down the ramp, no matter what the coefficient of friction is.*

of friction is high enough, the block should be brought to rest on the ramp, the static friction force opposing the force of gravity. Again, the contact forces do no work once the bodies are stationary. At most collisions, the response algorithm described thus far will bring the tangential velocity at the contact point to zero. As the collision ends, the block again becomes airborne, following a ballistic trajectory. During this interval, gravity is unchecked and will impart a linear acceleration with a component down the ramp. Upon the next collision, the downward velocity will again be halted, but the block motion during the ballistic phase remains uncorrected. Since the velocity of the block down the ramp varies between 0 and some small positive ε, the block will creep slowly down the ramp, much as a real block would if placed on a vibrating ramp.

To correct this problem, a second type of microcollision is employed. When the initial relative normal velocity satisfies (3.18), the unique impulse $\mathbf{p}_r$ that reverses this velocity is computed. This impulse is given by (3.5), with $\Delta\mathbf{u} = -2\mathbf{u}(\gamma_0)$:

$$\mathbf{p}_r = -2\mathbf{K}^{-1}\mathbf{u}(\gamma_0).$$

The cost of computing $\mathbf{p}_r$ is much less than that of performing a standard collision integration. Furthermore, by Corollary 1, $\mathbf{p}_r$ does no net work on the colliding bodies, like the static force is models. If $\mathbf{p}_r$ lies within the friction cone it is applied, otherwise the impulse is computed through collision integration with a boosted coefficient of restitution. The reversing microcollisions give the block a small "kick" back up the ramp, canceling the effects of gravity during the ballistic phase. Creep is still present, but is very small (Figure 3.15).

The decision to classify a collision as a microcollision is based only on local information at the contact point. This is in keeping with the impulse-based approach of not explicitly classifying macroscopic state, but instead generating correct macroscopic behavior through collisions. When one object settles on another, the process is a smooth transition from a situation in which virtually all collisions are ordinary collisions to a situation in which virtually all collisions are microcollisions. Having to make explicit decisions about when an object is resting on another, or whether a contact is of a rolling or sliding nature, creates artificially sharp transitions in system behavior, and leads to the consistency problems that arise in constraint-based simulation.

The ordinary collision response computations derived in this chapter are on a solid theoretical framework, given the assumptions made. Microcollisions are harder to justify

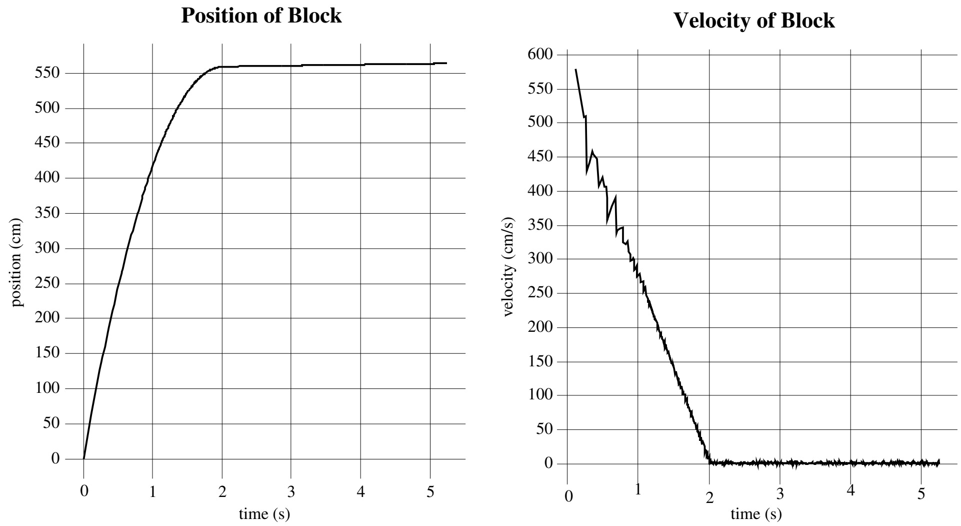

Figure 3.15: *Data from the block on ramp experiment using* Impulse. *A block is launched down a* 30° *ramp with an initial sliding velocity of* 548 *cm/s, with* $\mu = 0.9$ *and* $g = 981$ *cm/s*^{2}*. Theoretically, the block should experience constant deceleration for* 2.0 *seconds, coming to rest after traveling* 548 *cm. The simulator's results agree closely, although the plots reveal slight creep in the position of the block.*

from a physical standpoint. Important properties are preserved when using microcollisions; for example, the contact forces on the bodies still lie within friction cones, even when microcollisions are used. But ultimately microcollisions are an ad-hoc way to make impulse-based simulation feasible for modeling prolonged contact between bodies. There are limitations of what can be done, even using microcollisions. The creeping problem, although slight, may be unacceptable for some applications. The static contact between a book and a table poses no problem, nor does a stack of two books. By three books, the simulation is noticeably slower, and four or more causes it to grind to a halt. Too many collision impulses must be propagated up and down the stack of books, and the weight of the other books on the bottom book causes deep envelope penetration between the bottom book and the table. This penetration could be reduced by increasing e_{def}, but too high a value adds instability to ordinary situations. Furthermore, parameter tuning has been eschewed through the development of *Impulse*. A stack of books on a table is a situation which is much more naturally and efficiently handled with a constraint-based approach.

Chapter 4

Constrained Body Dynamics

Impulse-based simulation can not efficiently model tightly constrained contact, such as that occurring at a hinge joint. It is useful, however, when combined with a classical method for modeling this constrained contact. This combination is called *hybrid simulation*, and is discussed in Chapter 5. The present chapter develops the forward dynamics algorithms for several classes of *multibodies*: collections of rigid bodies connected by joints. The algorithms presented here are the original work of Featherstone [Fea83].

Featherstone's algorithm is a generalized coordinate approach, meaning that there are as many state variables as degrees of freedom in the system. The advantage of this approach is that there are no invalid state configurations; the constraints are automatically enforced. Maximal coordinate methods, also called multiplier methods, can also be used to derive the dynamics of constrained body systems. These employ more state variables than there are degrees of freedom, and so constraints must be continuously enforced. Multiplier methods are more naturally applied to more general kinematic structures, such as those with kinematic loops. They suffer, however, from drift problems due to the coordinate redundancy. Some differential-algebraic equation (DAE) solvers, such as MEXX [LNPE92], employ sophisticated methods for maintaining constraints, at the price of added complexity. A summary of the pros and cons of generalized and maximal coordinate approaches is in [Bar96]. Clear comparisons of the speed and accuracy of the two methods have not been published; the best choice may be application dependent.

The results of this chapter are not original, but are included for two reasons. The first is tutorial. Featherstone's algorithm has been presented by him as well as by several other dynamicists [BJO86, Lil93, JR], yet the algorithm remains confusing and difficult to implement for many who are not specialists in the area.[1] The sources mentioned above have a tendency to describe the algorithm from a very theoretical viewpoint, or to sacrifice detail and completeness for brevity. This chapter derives Featherstone's equations by taking a very different path: the equations are derived from the basic principles of rigid body dynamics, taught in any physics course. Spatial notation is adopted only after a thorough (non-spatial) treatment of the underlying quantities. Extensions of the basic serial chain algorithm to handle tree-like linkages and floating linkages are explicitly described. The second reason for this chapter is to provide a foundation for multibody dynamics which will be used in Chapter 5 to develop the collision response model for hybrid simulation.

Section 4.1 describes the basic forward dynamics problem for constrained linkages. Section 4.2 details how the absolute velocities of all links are determined from the joint positions and velocities, a procedure which is omitted in most expositions on constrained dynamics. Acceleration propagation equations are also derived. Section 4.3 introduces spatial algebra. Section 4.4 derives Featherstone's complete algorithm for grounded, serial linkages, from first principles of rigid body mechanics. Sections 4.5 and 4.6 extend the basic algorithm to tree-like linkages and floating linkages, respectively.

4.1 Constrained forward dynamics

Consider the serial linkage shown in Figure 4.1. The links are numbered from 1 to n, where link 1 is attached to a fixed base (called link 0), and link n is the outermost link. The joints are also labeled from 1 to n. For $1 \leq i < n$, link i has an inboard joint, which is closer to the base, and an outboard joint, which is closer to the other end of the linkage; link n has only an inboard joint. The inboard joint of link i is joint i ($1 \leq i \leq n$), and the outboard joint of link i is joint $i+1$ ($1 \leq i < n$). When each joint only allows one degree of freedom, a compact parameterization of the configuration space of the linkage is the vector of joint positions $\mathbf{q} = (q_1, \ldots, q_n)^T$. The problem to be solved is:

Problem 6 (Forward dynamics) Given: *The positions* $\mathbf{q}$ *and velocities* $\dot{\mathbf{q}}$ *of the* n *joints of a serial linkage, the external forces acting on the linkage, and the forces and torques being applied by the joint actuators.* Find: *The resulting accelerations of the joints,* $\ddot{\mathbf{q}}$.

[1] Baraff also makes this point in [Bar96].

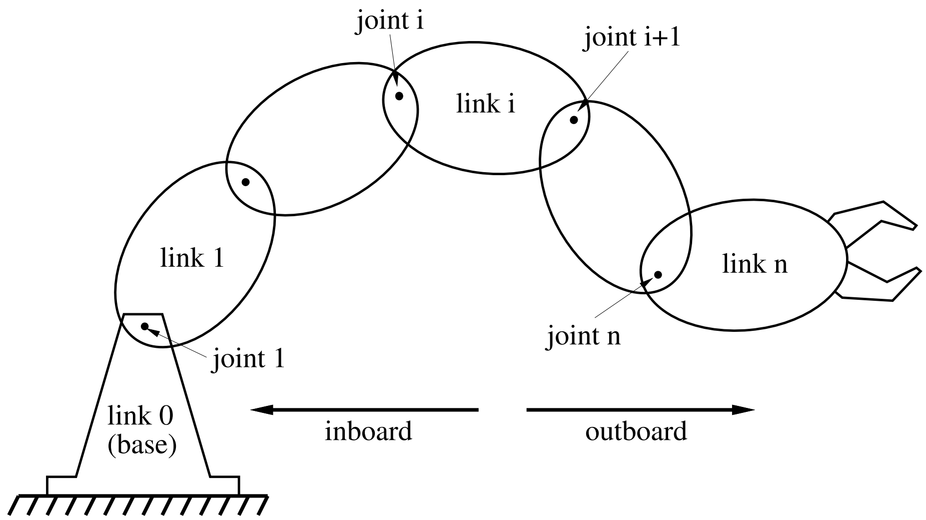

Figure 4.1: *The link and joint indexing conventions for serial linkages.*

A major hurdle in understanding the dynamics of linked rigid body systems is the complex notation that is required. For convenience, Table 4.1 summarizes the notation which is introduced and used throughout this chapter.

4.2 Velocity and acceleration propagation

To solve the forward dynamics problem one must first determine the absolute motion of all of the links of the linkage. The linear and angular velocities of the links depend only on $\mathbf{q}$ and $\dot{\mathbf{q}}$. The linear and angular accelerations of the links depend on these quantities and also on $\ddot{\mathbf{q}}$. Although $\ddot{\mathbf{q}}$ is initially unknown, the link accelerations can still be expressed in terms of it.

Definition 3 *Attached to each link i is a* **body frame** $\mathcal{F}_i$*, with origin at the link's center of mass, and axes aligned with the principle axes of inertia for the link. Vectors expressed in* **link coordinates** *have coordinates resolved relative to this body frame.*

The immediate goal is to solve this subproblem:

Problem 7 (Velocity and acceleration propagation) Given: *The joint positions* $\mathbf{q}$*, velocities,* $\dot{\mathbf{q}}$*, and accelerations* $\ddot{\mathbf{q}}$. Compute: *For each link, the linear velocity* $\mathbf{v}_i$*, angular velocity* $\boldsymbol{\omega}_i$*, linear acceleration* $\mathbf{a}_i$*, and angular acceleration* $\boldsymbol{\alpha}_i$*, all relative to an inertial frame.*

n	number of links of serial linkage
$\mathbf{v}_i$	linear velocity of link i
$\boldsymbol{\omega}_i$	angular velocity of link i
$\mathbf{a}_i$	linear acceleration of link i
$\boldsymbol{\alpha}_i$	angular acceleration of link i
$\mathbf{v}_{\mathrm{rel}}$	relative linear velocity of link i [Definition 4]
$\boldsymbol{\omega}_{\mathrm{rel}}$	relative angular velocity of link i [Definition 4]
m_i	mass of link i
$\mathbf{M}_i$	matricized mass of link i [Equation 4.25]
$\mathbf{I}_i$	diagonalized (body frame) inertia tensor of link i [Appendix A.3]
$\mathbf{d}_i$	vector from link i inboard joint to link i c.o.m. [Figure 4.2]
$\mathbf{r}_i$	vector from link $i-1$ c.o.m. to link i c.o.m. [Figure 4.2]
$\mathbf{u}_i$	unit vector in direction of joint i axis [Figure 4.2]
$\hat{\mathbf{I}}_i$	spatial isolated inertia of link i [Definition 13]
$\hat{\mathbf{I}}_i^A$	spatial articulated inertia of link i [Theorem 18]
$\hat{\mathbf{Z}}_i$	spatial isolated zero-acceleration (z.a.) force of link i [Definition 13]
$\hat{\mathbf{Z}}_i^A$	spatial articulated zero-acceleration (z.a.) force of link i [Theorem 18]
$\hat{\mathbf{f}}_i^I$	spatial force applied by inboard joint to link i [Definitions 7, 12]
$\hat{\mathbf{f}}_i^O$	spatial force applied by outboard joint to link i [Definitions 7, 12]
$\hat{\mathbf{v}}_i$	spatial velocity of link i [Definition 5]
$\hat{\mathbf{a}}_i$	spatial acceleration of link i [Definitions 5, 12]
$\hat{\mathbf{s}}_i$	spatial joint axis of joint i [Definition 9]
$\hat{\mathbf{c}}_i$	spatial Coriolis force for link i [Definition 10]
q_i	scalar position of joint i
$\dot{q}_i$	scalar velocity of joint i
$\ddot{q}_i$	scalar acceleration of joint i
$\boldsymbol{\nu}_i$	vector velocity of joint i [Equation 4.5]
$\boldsymbol{\xi}_i$	vector acceleration of joint i [Equation 4.6]
$\mathcal{O}$	inertial reference frame
$\mathcal{F}_i$	body frame of link i [Definition 3]
${}_{\mathcal{G}}\hat{\mathbf{X}}_{\mathcal{F}}$	spatial transformation from frame $\mathcal{F}$ to frame $\mathcal{G}$ [Definition 6]
${}_j\hat{\mathbf{X}}_i$	spatial transformation from frame $\mathcal{F}_i$ to frame $\mathcal{F}_j$

Table 4.1: *Notation used in Chapter 4*

The quantities $\mathbf{v}_i, \boldsymbol{\omega}_i, \mathbf{a}_i$, and $\boldsymbol{\alpha}_i$ describe the motion of frame $\mathcal{F}_i$ relative to an inertial frame $\mathcal{O}$. The strategy for solving this problem is straightforward. The velocities and accelerations of link i are completely determined by the velocities and accelerations of link $i-1$, and the motion of joint i. Since the velocities and accelerations of the base link are **0**, the base link is a starting point, and one proceeds inductively from there.

A few additional vectors are needed, as shown in Figure 4.2. Let $\mathbf{u}_i$ denote the

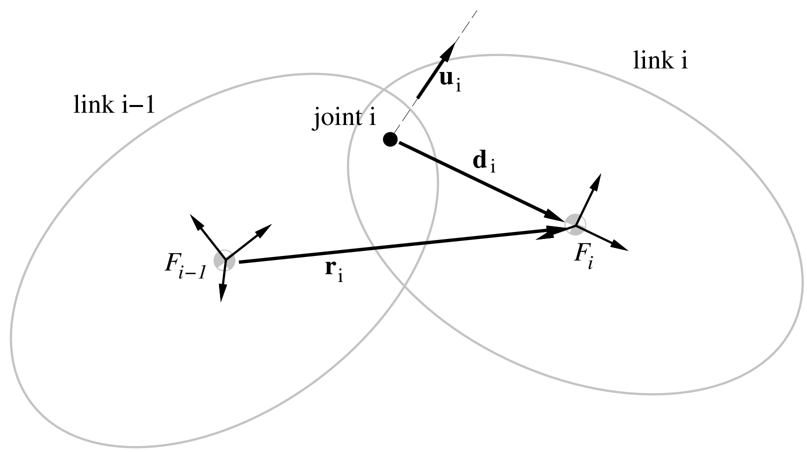

Figure 4.2: *Quantities for propagating velocities and accelerations.*

unit vector in the direction of the axis of joint i, $\mathbf{r}_i$ denote the vector from the origin of $\mathcal{F}_{i-1}$ to the origin of $\mathcal{F}_i$, and $\mathbf{d}_i$ denote some vector from the axis of joint i to the origin of $\mathcal{F}_i$.

The motion of link i may be divided into two parts: one due to the motion of link $i-1$, and one due to the motion of joint i. The latter component is called the relative motion.

Definition 4 (Relative velocities) *The* **linear relative velocity $\mathbf{v}_{rel}$** *is the linear center of mass velocity of link i due only to the motion of joint i, that is, the linear center of mass velocity that link i would have if link $i-1$ were held fixed. The* **angular relative velocity $\boldsymbol{\omega}_{rel}$** *is the angular velocity of link i due only do the motion of joint i, that is, the angular velocity that link i would have if link $i-1$ were held fixed.*

The angular velocity of link i is the angular velocity of link $i-1$ plus the angular velocity induced by rotation at the joint:

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{i-1} \ + \ \boldsymbol{\omega}_{\mathrm{rel}}. \tag{4.1}$$

Propagating linear velocity is similar, but there is an added term due to the rotation of link $i-1$:

$$\mathbf{v}_i = \mathbf{v}_{i-1} \ + \ \boldsymbol{\omega}_{i-1} \times \mathbf{r}_i \ + \ \mathbf{v}_{\text{rel}}. \tag{4.2}$$

The formula for propagating angular acceleration is obtained by differentiating (4.1):

$$\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i-1} \ + \ \dot{\boldsymbol{\omega}}_{\text{rel}}. \tag{4.3}$$

The formula for propagating linear acceleration is obtained by differentiating (4.2):

$$\mathbf{a}_i = \mathbf{a}_{i-1} \ + \ \boldsymbol{\alpha}_{i-1} \times \mathbf{r}_i \ + \ \boldsymbol{\omega}_{i-1} \times \dot{\mathbf{r}}_i \ + \ \dot{\mathbf{v}}_{\text{rel}}.$$

Since $\dot{\mathbf{r}}_i = \mathbf{v}_i - \mathbf{v}_{i-1}$, which by (4.2) is $\boldsymbol{\omega}_{i-1} \times \mathbf{r}_i + \mathbf{v}_{\text{rel}}$, the above equation can be rewritten

$$\mathbf{a}_i = \mathbf{a}_{i-1} \ + \ \boldsymbol{\alpha}_{i-1} \times \mathbf{r}_i \ + \ \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{r}_i) \ + \ \boldsymbol{\omega}_{i-1} \times \mathbf{v}_{\text{rel}} \ + \ \dot{\mathbf{v}}_{\text{rel}}. \tag{4.4}$$

Equations (4.1–4.4) express link i's motion in terms of link $i-1$'s motion, and the relative motion due to joint i. All that remains is to define $\boldsymbol{\omega}_{\text{rel}}$, $\mathbf{v}_{\text{rel}}$, and their time derivatives for prismatic and revolute joints. To this end, define two vectors along the axis of joint i:

$$\boldsymbol{\nu}_i = \dot{q}_i \mathbf{u}_i \tag{4.5}$$

$$\boldsymbol{\xi}_i = \ddot{q}_i \mathbf{u}_i. \tag{4.6}$$

Suppose joint i is prismatic. Alone it imparts no angular velocity to link i, only a linear center of mass velocity. From Figure 4.2 and Definition 4,

$$\boldsymbol{\omega}_{\text{rel}} = \mathbf{0} \tag{4.7}$$

$$\mathbf{v}_{\text{rel}} = \boldsymbol{\nu}_i. \tag{4.8}$$

On the other hand, a revolute joint i imparts both an angular velocity and a linear center of mass velocity to link i. From Figure 4.2 and Definition 4,

$$\boldsymbol{\omega}_{\text{rel}} = \boldsymbol{\nu}_i \tag{4.9}$$

$$\mathbf{v}_{\text{rel}} = \boldsymbol{\nu}_i \times \mathbf{d}_i. \tag{4.10}$$

The following lemma is useful in computing $\dot{\boldsymbol{\omega}}_{\text{rel}}$ and $\dot{\mathbf{v}}_{\text{rel}}$.

Lemma 6 *For prismatic or revolute joints,*

$$\dot{\boldsymbol{\nu}}_i = \boldsymbol{\xi}_i \ + \ \boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i.$$

Furthermore, for revolute joints,

$$\frac{d}{dt}(\boldsymbol{\nu}_i \times \mathbf{d}_i) = \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) \ + \ \boldsymbol{\xi}_i \times \mathbf{d}_i \ + \ \boldsymbol{\nu}_i \times (\boldsymbol{\nu}_i \times \mathbf{d}_i).$$

Proof: Differentiating (4.5),

$$\dot{\boldsymbol{\nu}}_i = \ddot{q}_i \mathbf{u}_i + \dot{q}_i \dot{\mathbf{u}}_i = \boldsymbol{\xi}_i + \dot{q}_i \dot{\mathbf{u}}_i.$$

Since the joint axis $\mathbf{u}_i$ rotates with link $i-1$, $\dot{\mathbf{u}} = \boldsymbol{\omega}_{i-1} \times \mathbf{u}_i$. Substituting this result into the above equation proves the first claim of the lemma. For the second claim, note

$$\frac{d}{dt}(\boldsymbol{\nu}_i \times \mathbf{d}_i) = \dot{\boldsymbol{\nu}}_i \times \mathbf{d}_i + \boldsymbol{\nu}_i \times \dot{\mathbf{d}}_i. \tag{4.11}$$

Now $\dot{\boldsymbol{\nu}}_i$ is given by the first claim, and $\mathbf{d}_i$ rotates with link i,

$$\dot{\mathbf{d}}_i = \boldsymbol{\omega}_i \times \mathbf{d}_i = (\boldsymbol{\omega}_{i-1} + \boldsymbol{\nu}_i) \times \mathbf{d}_i.$$

[The second equality above follows from (4.1) and (4.9).] Substituting for $\dot{\boldsymbol{\nu}}_i$ and $\dot{\mathbf{d}}_i$ into (4.11), distributing the cross products, and applying the identity

$$\mathbf{A} \times (\mathbf{B} \times \mathbf{C}) \ + \ \mathbf{B} \times (\mathbf{C} \times \mathbf{A}) = -\mathbf{C} \times (\mathbf{A} \times \mathbf{B})$$

proves the second claim. □

The results of Equations (4.7–4.10) plus Lemma 6 are summarized below.

	joint i prismatic	joint i revolute
$\boldsymbol{\omega}_{\text{rel}}$	$\mathbf{0}$	$\boldsymbol{\nu}_i$
$\mathbf{v}_{\text{rel}}$	$\boldsymbol{\nu}_i$	$\boldsymbol{\nu}_i \times \mathbf{d}_i$
$\dot{\boldsymbol{\omega}}_{\text{rel}}$	$\mathbf{0}$	$\boldsymbol{\xi}_i \ + \ \boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i$
$\dot{\mathbf{v}}_{\text{rel}}$	$\boldsymbol{\xi}_i \ + \ \boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i$	$\boldsymbol{\omega}_{i-1} \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) \ + \ \boldsymbol{\xi}_i \times \mathbf{d}_i \ + \ \boldsymbol{\nu}_i \times (\boldsymbol{\nu}_i \times \mathbf{d}_i)$

Substituting these quantities into equations (4.1–4.4) produce the propagation formulae that give link i's motion in terms of link $i-1$'s motion, and the motion of joint i. If joint i is prismatic,

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{i-1} \tag{4.12}$$

$$\mathbf{v}_i = \mathbf{v}_{i-1} \ + \ \boldsymbol{\omega}_{i-1} \times \mathbf{r}_i \ + \ \boldsymbol{\nu}_i \tag{4.13}$$

$$\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i-1} \tag{4.14}$$

$$\mathbf{a}_i = \mathbf{a}_{i-1} \ + \ \boldsymbol{\alpha}_{i-1} \times \mathbf{r}_i \ + \ \boldsymbol{\xi}_i \ + \ \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{r}_i) \ + \ 2\boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i. \tag{4.15}$$

If joint i is revolute,

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{i-1} + \boldsymbol{\nu}_i \tag{4.16}$$

$$\mathbf{v}_i = \mathbf{v}_{i-1} + \boldsymbol{\omega}_{i-1} \times \mathbf{r}_i + \boldsymbol{\nu}_i \times \mathbf{d}_i \tag{4.17}$$

$$\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i-1} + \boldsymbol{\xi}_i + \boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i \tag{4.18}$$

$$\begin{aligned}\mathbf{a}_i = {} & \mathbf{a}_{i-1} + \boldsymbol{\alpha}_{i-1} \times \mathbf{r}_i + \boldsymbol{\xi}_i \times \mathbf{d}_i \\ & + \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{r}_i) + 2\boldsymbol{\omega}_{i-1} \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) + \boldsymbol{\nu}_i \times (\boldsymbol{\nu}_i \times \mathbf{d}_i).\end{aligned} \tag{4.19}$$

In Equations (4.12–4.19), all velocity and acceleration vectors are absolute, meaning they are taken relative to the inertial frame $\mathcal{O}$. In computations, however, the coordinates of the vectors may be expressed relative to any convenient coordinate frame. Computations are simplified if all vectors are expressed in $\mathcal{F}_i$, the body frame attached to link i.

As an example, consider the planar linkage of Figure 4.3. Assuming joint 1

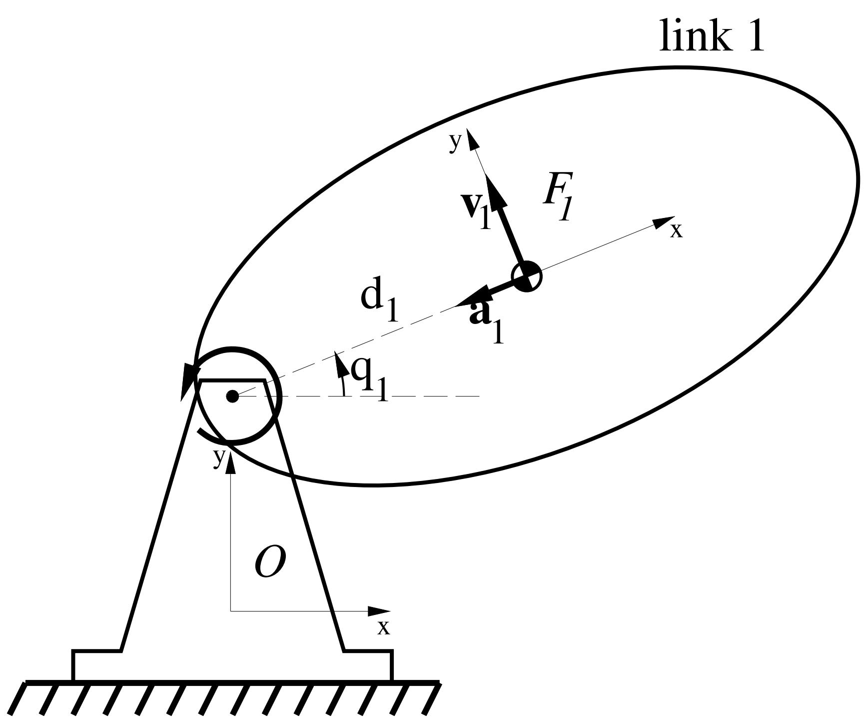

Figure 4.3: *The vectors* $\mathbf{v}_1$ *and* $\mathbf{a}_1$ *describe the motion of the center of mass of link* 1 *relative to the inertial frame* $\mathcal{O}$. *But the coordinates of these vectors may be expressed in any frame.*

rotates counter-clockwise with constant angular velocity, the expressions for $\mathbf{v}_1$ and $\mathbf{a}_1$ in

$\mathcal{O}$'s coordinates are

$$\mathbf{v}_1 = \begin{bmatrix} -\dot{q}_1 d_1 \sin q_1 \\ \dot{q}_1 d_1 \cos q_1 \end{bmatrix} \qquad \mathbf{a}_1 = \begin{bmatrix} -\dot{q}_1^2 d_1 \cos q_1 \\ -\dot{q}_1^2 d_1 \sin q_1 \end{bmatrix}.$$

The same vectors expressed in $\mathcal{F}_i$'s coordinates are

$$\mathbf{v}_1 = \begin{bmatrix} 0 \\ \dot{q}_1 d_1 \end{bmatrix} \qquad \mathbf{a}_1 = \begin{bmatrix} -\dot{q}_1^2 d_1 \\ 0 \end{bmatrix}.$$

These latter expressions are simpler because $\mathcal{F}_1$ rotates with link 1. Also, the vectors $\mathbf{d}_i$ and $\mathbf{u}_i$ are fixed in $\mathcal{F}_i$, so many of the subexpressions appearing in (4.12–4.19) may be precomputed. The only catch is that vector coordinates expressed in $\mathcal{F}_{i-1}$ must be transformed to frame $\mathcal{F}_i$ before they can be used in computations with other vectors expressed in $\mathcal{F}_i$. This amounts to multiplying the four vectors $\boldsymbol{\omega}_{i-1}, \mathbf{v}_{i-1}, \boldsymbol{\alpha}_{i-1}$, and $\mathbf{a}_{i-1}$ by a rotation matrix before using them in (4.12–4.19). If joint i is prismatic, this rotation matrix is constant and may also be precomputed; if join i is revolute, the matrix depends on q_i.

The first step in the forward dynamics algorithm is the computation of the absolute linear and angular velocities of all links. These are determined from the known dynamic state $(\mathbf{q}, \dot{\mathbf{q}})$, and determine the centripetal and Coriolis forces acting on the links. The algorithm for computing these velocities follows directly from Equations (4.12)–(4.13) and (4.16)–(4.17), and is shown in Figure 4.4. The derived acceleration propagation equations involve the unknown joint accelerations through the term $\boldsymbol{\xi}_i$. These will be be needed in deriving the rest of the forward dynamics algorithm.

4.3 Spatial algebra

Spatial algebra is a powerful notation for describing quantities and relations relevant to the dynamics of three-dimensional systems. A spatial vector is six-dimensional and replaces two ordinary vectors in $\mathbb{R}^3$. The spatial acceleration of a rigid body describes both its linear and angular accelerations. The spatial force applied to a rigid body describes both the applied (ordinary) 3-D force and the applied 3-D moment. Throughout this chapter, Featherstone's convention of denoting spatial quantities with a caret is used.

Definition 5 *For a rigid body moving through space, with a frame $\mathcal{F}$ attached to it, let $\mathbf{v}$, $\boldsymbol{\omega}$, $\mathbf{a}$, and $\boldsymbol{\alpha}$ be the linear velocity, angular velocity, linear acceleration, and angular*

```
compSerialLinkVelocities

    ω0, v0, α0, a0 ← 0

    for i = 1 to n
        R ← rotation matrix from F_{i−1} to F_i
        r ← radius vector from F_{i−1} to F_i (in F_i coordinates)
        ω_i ← R ω_{i−1}
        v_i ← R v_{i−1}  +  ω_i × r
        if joint i is prismatic
            v_i ← v_i + q̇_i u_i
        else /* joint i is revolute */
            ω_i ← ω_i + q̇_i u_i
            v_i ← v_i + q̇_i (u_i × d_i)
```

Figure 4.4: `compSerialLinkVelocities`. Compute the absolute linear and angular velocities of all links of a serial linkage, in link coordinates. Input data are the dynamic state $(\mathbf{q}, \dot{\mathbf{q}})$. Output data are the velocities $\boldsymbol{\omega}_i$ and $\mathbf{v}_i$ for each link.

acceleration of frame $\mathcal{F}$ relative to an inertial frame $\mathcal{O}$. Then the **spatial velocity** $\hat{\mathbf{v}}$ *and* **spatial acceleration** $\hat{\mathbf{a}}$ *of the body, expressed in frame $\mathcal{F}$, are given by*

$$\hat{\mathbf{v}} = \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v} \end{bmatrix} \qquad \hat{\mathbf{a}} = \begin{bmatrix} \boldsymbol{\alpha} \\ \mathbf{a} \end{bmatrix}.$$

Now consider a rigid body moving through space with two frames, $\mathcal{F}$ and $\mathcal{G}$ attached to it, where $\mathcal{G}$ is displaced but not rotated relative to $\mathcal{F}$. Let $\mathbf{v}_{\mathcal{F}}$ and $\mathbf{v}_{\mathcal{G}}$ be the linear velocities of the origins of these frames, let $\boldsymbol{\omega}$ be the common angular velocity of the frames, and let $\mathbf{r}$ be the offset vector from the origin of $\mathcal{F}$ to the origin of $\mathcal{G}$. Then $\mathbf{v}_{\mathcal{G}} = \mathbf{v}_{\mathcal{F}} + \boldsymbol{\omega} \times \mathbf{r}$, and so

$$\hat{\mathbf{v}}_{\mathcal{G}} = \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v}_{\mathcal{G}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v}_{\mathcal{F}} + \boldsymbol{\omega} \times \mathbf{r} \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ -\tilde{\mathbf{r}} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v}_{\mathcal{F}} \end{bmatrix}. \tag{4.20}$$

($\mathbf{1}$ is the 3×3 identity matrix; see Appendix A.2 for a definition of $\tilde{\mathbf{r}}$.) The matrix on the right hand side is a 6×6 *spatial matrix*. If $\mathcal{G}$ is also rotated relative to $\mathcal{F}$, and $\mathbf{R}$ is the rotation matrix that transforms vector coordinates in $\mathcal{F}$ to vector coordinates in $\mathcal{G}$, the vector on the right hand side must be premultiplied by the spatial matrix

$$\begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}$$

in order to account for the rotation of $\mathcal{G}$ relative to $\mathcal{F}$. This leads to the following definition.

Definition 6 *Let $\mathcal{F}$ and $\mathcal{G}$ be two frames, let $\mathbf{r}$ be the offset vector from the origin of $\mathcal{F}$ to the origin of $\mathcal{G}$ (expressed in $\mathcal{G}$'s coordinates), and let $\mathbf{R}$ be the 3×3 rotation matrix transforming (non-spatial) vectors from $\mathcal{F}$ to $\mathcal{G}$. Then the 6×6* **spatial transformation matrix** *from $\mathcal{F}$ to $\mathcal{G}$ is given by*

$$_{\mathcal{G}}\hat{\mathbf{X}}_{\mathcal{F}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ -\tilde{\mathbf{r}} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ -\tilde{\mathbf{r}}\mathbf{R} & \mathbf{R} \end{bmatrix}.$$

With this notation, the spatial velocity transformation shown above can be written

$$\hat{\mathbf{v}}_{\mathcal{G}} = {_{\mathcal{G}}\hat{\mathbf{X}}_{\mathcal{F}}}\ \hat{\mathbf{v}}_{\mathcal{F}}.$$

A spatial transformation matrix is the analog of an ordinary rotation matrix: it transforms the coordinates of a spatial vector from one frame to another. The matrix $_{\mathcal{G}}\hat{\mathbf{X}}_{\mathcal{F}}$ transforms spatial accelerations from frame $\mathcal{F}$ to frame $\mathcal{G}$ in the same way as it transforms velocities. Spatial vectors are also useful for describing external influences acting on rigid bodies. Given a rigid body with an attached frame $\mathcal{F}$, one can express the total external influences on the body as a force with line of action passing through the origin of $\mathcal{F}$, and a torque.

Definition 7 *For a rigid body with attached frame $\mathcal{F}$, let the total external influences on the body be given by a force $\mathbf{f}$ with line of action passing through the origin of $\mathcal{F}$, and a torque $\boldsymbol{\tau}$. Then the* **spatial force** *acting on the body, expressed in frame $\mathcal{F}$ is*

$$\hat{\mathbf{f}} = \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} \end{bmatrix}.$$

Let $\mathcal{G}$ be a second frame attached to the rigid body, and let $\mathbf{r}$ be the offset vector from the origin of $\mathcal{F}$ to the origin of $\mathcal{G}$. The force acting through the origin of $\mathcal{F}$ may be displaced to act through the origin of $\mathcal{G}$, if a correction torque of $-\mathbf{r}\times\mathbf{f}$ is added; the torque $\boldsymbol{\tau}$ requires no adjustment (see [MK86] for details). Therefore,

$$\hat{\mathbf{f}}_{\mathcal{G}} = \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau}_{\mathcal{G}} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau}_{\mathcal{F}} \; - \; \mathbf{r}\times\mathbf{f} \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ -\tilde{\mathbf{r}} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau}_{\mathcal{F}} \end{bmatrix}.$$

The parallel to (4.20) is obvious; the same spatial transformation matrix ${}_{\mathcal{G}}\hat{\mathbf{X}}_{\mathcal{F}}$ that transforms spatial accelerations from frame $\mathcal{F}$ to frame $\mathcal{G}$, also transforms spatial forces from frame $\mathcal{F}$ to frame $\mathcal{G}$.

Definition 8 *Let*

$$\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}$$

be the coordinate representation of a spatial vector, with $\mathbf{a},\mathbf{b}\in\mathbb{R}^3$. *The* **spatial transpose** *of* $\hat{\mathbf{x}}$*, denoted by* $\hat{\mathbf{x}}'$*, is*

$$\hat{\mathbf{x}}' = \left[\mathbf{b}^T, \mathbf{a}^T\right].$$

The **spatial inner product** *of two spatial vectors* $\hat{\mathbf{x}}$ *and* $\hat{\mathbf{y}}$ *is given by* $\hat{\mathbf{x}}'\hat{\mathbf{y}}$*.*

To transpose a spatial vector, one swaps its two halves and transposes each of them; this is typically what one wants. For example, if $\hat{\mathbf{v}}$ is the spatial velocity of a body, and $\hat{\mathbf{f}}$ is a spatial force applied to the body, the inner product of spatial force and spatial velocity is

$$\hat{\mathbf{f}}'\hat{\mathbf{v}} = \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} \end{bmatrix}' \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v} \end{bmatrix} = \left[\boldsymbol{\tau}^T, \mathbf{f}^T\right] \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v} \end{bmatrix} = \boldsymbol{\tau}\cdot\boldsymbol{\omega} \; + \; \mathbf{f}\cdot\mathbf{v},$$

which is power, as with ordinary 3-D vectors.

4.3.1 Spatial formulation of acceleration propagation

The acceleration propagation equations developed in Section 4.2 are compactly expressed with spatial algebra. If joint i is prismatic, equations (4.14)–(4.15) can be written

$$\begin{bmatrix} \boldsymbol{\alpha}_i \\ \mathbf{a}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}_{i-1} \\ -\mathbf{r}_i\times\boldsymbol{\alpha}_{i-1} \; + \; \mathbf{a}_{i-1} \end{bmatrix} + \ddot{q}_i \begin{bmatrix} \mathbf{0} \\ \mathbf{u}_i \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\omega}_{i-1}\times(\boldsymbol{\omega}_{i-1}\times\mathbf{r}_i) + 2\boldsymbol{\omega}_{i-1}\times\boldsymbol{\nu}_i \end{bmatrix}. \tag{4.21}$$

If joint i is revolute, equations (4.18)–(4.19) can be written

$$\begin{bmatrix} \boldsymbol{\alpha}_i \\ \mathbf{a}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}_{i-1} \\ -\mathbf{r}_i \times \boldsymbol{\alpha}_{i-1} + \mathbf{a}_{i-1} \end{bmatrix} + \ddot{q}_i \begin{bmatrix} \mathbf{u}_i \\ \mathbf{u}_i \times \mathbf{d}_i \end{bmatrix} + \begin{bmatrix} \boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i \\ \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{r}_i) + 2\boldsymbol{\omega}_{i-1} \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) + \boldsymbol{\nu}_i \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) \end{bmatrix}. \quad (4.22)$$

Equations (4.21) and (4.22)are unified with the help of the following definitions.

Definition 9 *The* **spatial joint axis** *of joint i is the spatial vector*

$$\hat{\mathbf{s}}_i = \begin{bmatrix} \mathbf{0} \\ \mathbf{u}_i \end{bmatrix}$$

if the joint is prismatic, and the spatial vector

$$\hat{\mathbf{s}}_i = \begin{bmatrix} \mathbf{u}_i \\ \mathbf{u}_i \times \mathbf{d}_i \end{bmatrix}$$

if the joint is revolute.

Definition 10 *The* **spatial Coriolis force** *of link i is the spatial vector*

$$\hat{\mathbf{c}}_i = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{r}_i) + 2\boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i \end{bmatrix}$$

if the inboard joint i is prismatic, and the spatial vector

$$\hat{\mathbf{c}}_i = \begin{bmatrix} \boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i \\ \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{r}_i) + 2\boldsymbol{\omega}_{i-1} \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) + \boldsymbol{\nu}_i \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) \end{bmatrix}$$

if the inboard joint i is revolute.

The first vector on the right hand sides of (4.21) and (4.22) is just the spatial acceleration of link $i-1$ transformed to frame $\mathcal{F}_i$. Using the shorthand ${}_j\hat{\mathbf{X}}_i$ for ${}_{\mathcal{F}_j}\hat{\mathbf{X}}_{\mathcal{F}_i}$, this vector can be written ${}_i\hat{\mathbf{X}}_{i-1}\hat{\mathbf{a}}_{i-1}$. Thus, (4.21) and (4.22) can both be written as

$$\hat{\mathbf{a}}_i = {}_i\hat{\mathbf{X}}_{i-1}\hat{\mathbf{a}}_{i-1} + \ddot{q}_i\hat{\mathbf{s}}_i + \hat{\mathbf{c}}_i. \quad (4.23)$$

This compact result encapsulates the acceleration propagation equations derived in Section 4.2. One advantage of the spatial form is that prismatic and revolute joints are put into a common framework; differences only appear in the underlying expressions for $\hat{\mathbf{s}}_i$ and $\hat{\mathbf{c}}_i$. By the established convention, vectors subscripted i are expressed in frame $\mathcal{F}_i$. Hence, $\hat{\mathbf{s}}_i$ is constant and may be precomputed. The vector $\hat{\mathbf{c}}_i$ comprises Coriolis and centripetal acceleration terms. It involves velocities of link $i-1$, which must be transformed into frame $\mathcal{F}_i$, and velocities of link i. It depends on the positions and velocities of joints 1 through i, but not on any joint accelerations.

4.4 The Featherstone algorithm

The Featherstone algorithm is a method of solving the forward dynamics problem in $O(n)$ time for an n link manipulator. It is one of a family of methods called *structurally recursive* algorithms. These methods are alternatives to the $O(n^3)$ algorithms that explicitly build the mass matrix for the system and must invert it to solve for joint accelerations [WO82].

Definition 11 (Articulated body of a serial linkage) *For a serial linkage with n links, sever the linkage at joint i, and consider the connected subchain comprising links $i, \ldots, n$ in isolation, detached from the fixed base. This subchain is called an* **articulated body** *of the original linkage, and link i is called the* **handle** *of the articulated body. The trivial articulated body with link n as a handle is simply the isolated rigid body of link n.*

Featherstone's algorithm works by considering successive articulated bodies of the original linkage, beginning with the trivial case of link n alone, and adding inboard links one by one. The key idea is to relate the spatial acceleration of the handle to the spatial force applied at its inboard joint. The links outboard to the handle will have an effect on this relation.

Definition 12 (Per link spatial vectors) *The spatial acceleration of link i is denoted $\hat{\mathbf{a}}_i$, the spatial force exerted on link i through its inboard joint is $\hat{\mathbf{f}}_i^I$, and the spatial force exerted on link i through its outboard joint is $\hat{\mathbf{f}}_i^O$.* The coordinates of all of these spatial vectors are expressed in frame $\mathcal{F}_i$.

Even though the inboard influences on link i are exerted at the joint i, the spatial force $\hat{\mathbf{f}}_i^I$ is given by $(\mathbf{f}_i^I, \boldsymbol{\tau}_i^I)^T$, where $\mathbf{f}_i^I$ and $\boldsymbol{\tau}_i^I$ are the equivalent force and torque applied at the origin of $\mathcal{F}_i$; the line of action of $\mathbf{f}_i^I$ passes through the origin of $\mathcal{F}_i$. The same is true for the outboard spatial force, $\mathbf{f}_i^O$.

Theorem 18 (Serial linkage articulated motion) *Consider the articulated body of a serial linkage that has link i as a handle ($1 \leq i \leq n$). There exists a spatial matrix $\hat{\mathbf{I}}_i^A$, and a spatial vector $\hat{\mathbf{Z}}_i^A$ such that*

$$\hat{\mathbf{f}}_i^I = \hat{\mathbf{I}}_i^A \hat{\mathbf{a}}_i + \hat{\mathbf{Z}}_i^A. \tag{4.24}$$

$\hat{\mathbf{I}}_i^A$ *is called the* **spatial articulated inertia** *of link i, and is independent of the joint velocities and accelerations.* $\hat{\mathbf{Z}}_i^A$ *is called the* **spatial articulated zero-acceleration (z.a.) force** *of link i, and is independent of the joint accelerations.*

The adjective *articulated* indicates that the entire subchain beginning at the handle is being considered, as opposed to the handle in isolation. $\hat{\mathbf{Z}}_i^A$ is so named since by (4.24) it is the force which must be exerted by the inboard joint on link i, if link i is not to accelerate ($\hat{\mathbf{a}}_i = \hat{\mathbf{0}}$).[2] The proof of Theorem 18 is involved, but constructive; an algorithm will follow directly from the proof. The proof is by induction on i, beginning with the trivial subchain comprising only the outermost link, and then adding links on the inboard side, one by one, until the entire linkage is considered.

4.4.1 Base case

For the base case of the induction, $i = n$, and the subchain being considered is simply the last link of the chain, as shown in Figure 4.5. This free body diagram depicts all of the forces and torques acting on the link: those applied by the inboard joint (but resolved at the origin of $\mathcal{F}_n$), and those due to external influences like gravity. The motion of the body is given by the Newton-Euler equations described in Appendix A.3. In link coordinates,

$$\begin{aligned} \mathbf{f}_n^I + m_n \mathbf{g} &= m_n \mathbf{a}_n \\ \boldsymbol{\tau}_n^I &= \mathbf{I}_n \boldsymbol{\alpha}_n + \boldsymbol{\omega}_n \times \mathbf{I}_n \boldsymbol{\omega}_n, \end{aligned}$$

where m_n is the mass of the link n, and $\mathbf{I}_n$ is the inertia tensor in link coordinates. These equations can be written as a single spatial equation. If

$$\mathbf{M}_i = m_i \mathbf{1}, \tag{4.25}$$

[2]Featherstone calls these spatial articulated z.a. forces *bias forces*, and denotes them $\hat{\mathbf{P}}_i$.

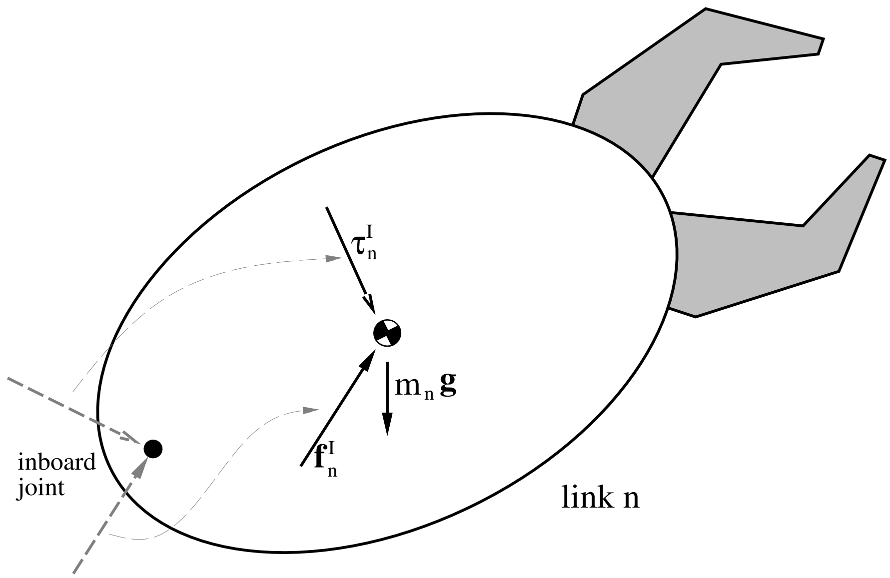

Figure 4.5: *The free body diagram of the last link of the serial linkage. It is the handle of a trivial subchain.*

then

$$\begin{bmatrix} \mathbf{f}_n^I \\ \boldsymbol{\tau}_n^I \end{bmatrix} = \begin{bmatrix} 0 & \mathbf{M}_n \\ \mathbf{I}_n & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_n \\ \mathbf{a}_n \end{bmatrix} + \begin{bmatrix} -m_n \mathbf{g} \\ \boldsymbol{\omega}_n \times \mathbf{I}_n \boldsymbol{\omega}_n \end{bmatrix},$$

or more compactly,

$$\hat{\mathbf{f}}_n = \hat{\mathbf{I}}_n^A \hat{\mathbf{a}}_n + \hat{\mathbf{Z}}_n^A.$$

Note that $\hat{\mathbf{Z}}_n^A$ is independent of the acceleration of link n, and therefore independent of the joint accelerations. $\hat{\mathbf{I}}_n^A$ is independent of the joint velocities and accelerations. In fact, it depends only on the mass properties of link n and may be precomputed. Thus Theorem 18 holds for the base case $i = n$. Two expressions above will be useful later, and are given special names.

Definition 13 *The* **spatial isolated inertia** *of link i is defined as the spatial matrix*

$$\hat{\mathbf{I}}_i = \begin{bmatrix} \mathbf{0} & \mathbf{M}_i \\ \mathbf{I}_i & \mathbf{0} \end{bmatrix}.$$

The **spatial isolated zero-acceleration (z.a.) force** *of link i is defined as the spatial vector*

$$\hat{\mathbf{Z}}_i = \begin{bmatrix} -m_i\mathbf{g} \\ \boldsymbol{\omega}_i \times \mathbf{I}_i\boldsymbol{\omega}_i \end{bmatrix}.$$

These two quantities are similar to their articulated counterparts, $\hat{\mathbf{I}}_i^A$ and $\hat{\mathbf{Z}}_i^A$, except that $\hat{\mathbf{I}}_i$ and $\hat{\mathbf{Z}}_i$ apply to link i in isolation, with no outboard links attached. For the special case of $i = n$, the definitions coincide: $\hat{\mathbf{I}}_i^A = \hat{\mathbf{I}}_i$ and $\hat{\mathbf{Z}}_i^A = \hat{\mathbf{Z}}_i$.

4.4.2 Inductive case

Assume Theorem 18 is true for i, and consider link $i-1$ (Figure 4.6). In addition

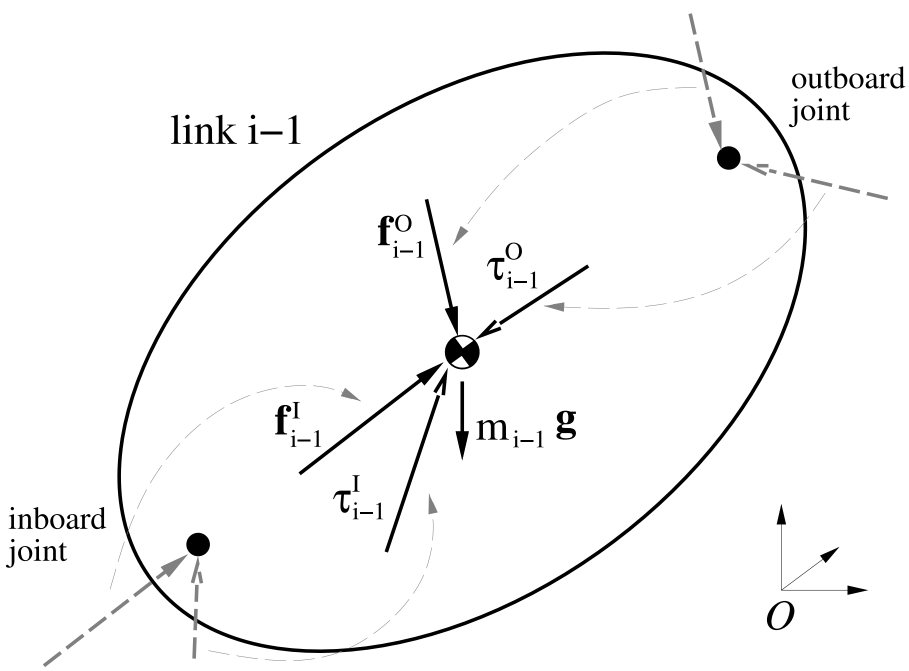

Figure 4.6: *The free body diagram of an inner link of the serial linkage. It is the handle of a non-trivial subchain.*

to gravity, and the force and torque applied through the inboard joint, there are also the force and torque applied through the outboard joint, $\mathbf{f}_{i-1}^O$ and $\boldsymbol{\tau}_{i-1}^O$. As before, all forces and torques are resolved at the origin of the link frame. Again using the Newton-Euler equations, the equations of motion of link $i-1$ are given by

$$\begin{bmatrix} \mathbf{f}_{i-1}^I \\ \boldsymbol{\tau}_{i-1}^I \end{bmatrix} + \begin{bmatrix} \mathbf{f}_{i-1}^O \\ \boldsymbol{\tau}_{i-1}^O \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{M}_{i-1} \\ \mathbf{I}_{i-1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_{i-1} \\ \mathbf{a}_{i-1} \end{bmatrix} + \begin{bmatrix} -m_{i-1}\mathbf{g} \\ \boldsymbol{\omega}_{i-1} \times \mathbf{I}_{i-1}\boldsymbol{\omega}_{i-1} \end{bmatrix}.$$

Using spatial notation and rearranging yields

$$\hat{\mathbf{f}}_{i-1}^I = \hat{\mathbf{I}}_{i-1}\hat{\mathbf{a}}_{i-1} + \hat{\mathbf{Z}}_{i-1} - \hat{\mathbf{f}}_{i-1}^O.$$

The force $\hat{\mathbf{f}}_{i-1}^O$ is the equal but opposite reaction force to $\hat{\mathbf{f}}_i^I$. Hence,

$$\hat{\mathbf{f}}_{i-1}^O = -\ {}_{i-1}\hat{\mathbf{X}}_i \hat{\mathbf{f}}_i^I.$$

Combining the above two equations,

$$\hat{\mathbf{f}}_{i-1}^I = \hat{\mathbf{I}}_{i-1}\hat{\mathbf{a}}_{i-1} + \hat{\mathbf{Z}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \hat{\mathbf{f}}_i^I.$$

Invoking the inductive hypothesis (4.24) for link i gives

$$\hat{\mathbf{f}}_{i-1}^I = \hat{\mathbf{I}}_{i-1}\hat{\mathbf{a}}_{i-1} + \hat{\mathbf{Z}}_{i-1} + \ {}_{i-1}\hat{\mathbf{X}}_i (\hat{\mathbf{I}}_i^A \hat{\mathbf{a}}_i + \hat{\mathbf{Z}}_i^A).$$

To put this equation in the form of (4.24), the acceleration $\hat{\mathbf{a}}_i$ must be eliminated from the right hand side. This is done by expressing $\hat{\mathbf{a}}_i$ in terms of $\hat{\mathbf{a}}_{i-1}$ using the acceleration propagation equation. Substituting (4.23) into the above equation and rearranging gives

$$\hat{\mathbf{f}}_{i-1}^I = \left(\hat{\mathbf{I}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \hat{\mathbf{I}}_i^A \, {}_i\hat{\mathbf{X}}_{i-1}\right)\hat{\mathbf{a}}_{i-1} + \hat{\mathbf{Z}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \left[\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A \hat{\mathbf{c}}_i + \left(\hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i\right)\ddot{q}_i\right]. \tag{4.26}$$

Now $\ddot{q}_i$ must be eliminated from the right hand side. The following lemma will help.

Lemma 7 *At every joint,*

$$Q_i = \hat{\mathbf{s}}_i' \hat{\mathbf{f}}_i^I,$$

where Q_i is the magnitude of the force exerted by the prismatic joint actuator, or the magnitude of the torque exerted by the revolute joint actuator.

Proof: Suppose a force $\mathbf{f}$ and a torque $\boldsymbol{\tau}$ are applied to link i *at the inboard joint.* These give rise to a spatial inboard force (resolved in the body frame) of

$$\hat{\mathbf{f}}_i^I = \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} - \mathbf{d}_i \times \mathbf{f} \end{bmatrix}.$$

If joint i is prismatic,

$$\hat{\mathbf{s}}_i'\hat{\mathbf{f}}_i^I = \begin{bmatrix} \mathbf{0} \\ \mathbf{u}_i \end{bmatrix}' \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} - \mathbf{d}_i \times \mathbf{f} \end{bmatrix} = \mathbf{f} \cdot \mathbf{u}_i.$$

The right hand side is the component of the applied force along the joint axis. This force must be supported by the actuator, hence, it is Q_i. If joint i is revolute,

$$\hat{\mathbf{s}}_i'\hat{\mathbf{f}}_i^I = \begin{bmatrix} \mathbf{u}_i \\ \mathbf{u} \times \mathbf{d}_i \end{bmatrix}' \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} - \mathbf{d}_i \times \mathbf{f} \end{bmatrix} = \mathbf{f} \cdot (\mathbf{u}_i \times \mathbf{d}_i) + (\boldsymbol{\tau} - \mathbf{d}_i \times \mathbf{f}) \cdot \mathbf{u}_i.$$

The right hand side reduces to $\boldsymbol{\tau} \cdot \mathbf{u}_i$, the component of the applied torque along the joint axis. This torque must be supported by the actuator, hence, it is Q_i. □

Substituting equation (4.23) for link i's spatial acceleration into (4.24) yields

$$\hat{\mathbf{f}}_i^I = \hat{\mathbf{I}}_i^A ({}_i\hat{\mathbf{X}}_{i-1}\hat{\mathbf{a}}_{i-1} + \ddot{q}_i\hat{\mathbf{s}}_i + \hat{\mathbf{c}}_i) + \hat{\mathbf{Z}}_i^A.$$

Premultiplying both sides by $\hat{\mathbf{s}}_i'$ and applying Lemma 7 gives

$$Q_i = \hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A ({}_i\hat{\mathbf{X}}_{i-1}\hat{\mathbf{a}}_{i-1} + \ddot{q}_i\hat{\mathbf{s}}_i + \hat{\mathbf{c}}_i) + \hat{\mathbf{s}}_i'\hat{\mathbf{Z}}_i^A,$$

from which $\ddot{q}_i$ may be determined:

$$\ddot{q}_i = \frac{Q_i - \hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\, {}_i\hat{\mathbf{X}}_{i-1}\hat{\mathbf{a}}_{i-1} - \hat{\mathbf{s}}_i'\left(\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i\right)}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}. \tag{4.27}$$

Substituting this expression for $\ddot{q}_i$ into (4.26) and rearranging gives

$$\begin{aligned}
\hat{\mathbf{f}}_{i-1}^I \quad = \quad & \left[\hat{\mathbf{I}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \left(\hat{\mathbf{I}}_i^A - \frac{\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}\right) {}_i\hat{\mathbf{X}}_{i-1}\right]\hat{\mathbf{a}}_{i-1} \\
& + \hat{\mathbf{Z}}_{i-1} \; + \; {}_{i-1}\hat{\mathbf{X}}_i \left[\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i + \frac{\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i\left[Q_i - \hat{\mathbf{s}}_i'\left(\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i\right)\right]}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}\right].
\end{aligned}$$

Comparing this to the desired form (4.24),

$$\hat{\mathbf{I}}_{i-1}^A \quad = \quad \hat{\mathbf{I}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \left(\hat{\mathbf{I}}_i^A - \frac{\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}\right) {}_i\hat{\mathbf{X}}_{i-1} \tag{4.28}$$

$$\hat{\mathbf{Z}}_{i-1}^A \quad = \quad \hat{\mathbf{Z}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \left[\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i + \frac{\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i\left[Q_i - \hat{\mathbf{s}}_i'\left(\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i\right)\right]}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}\right]. \tag{4.29}$$

These equations compute the articulated inertia and articulated z.a. force for link $i-1$ of the chain, given those same quantities for link i. $\hat{\mathbf{I}}_{i-1}$ and $\hat{\mathbf{s}}_i$ are constants, ${}_{i-1}\hat{\mathbf{X}}_i$ and ${}_i\hat{\mathbf{X}}_{i-1}$ depend only on joint positions, and $\hat{\mathbf{c}}_i$ depends only on joint positions and velocities. It follows that if $\hat{\mathbf{I}}_i^A$ is independent of joint velocities and accelerations, then so is $\hat{\mathbf{I}}_{i-1}^A$. Also, if $\hat{\mathbf{Z}}_i^A$ is independent of joint accelerations, then so is $\hat{\mathbf{Z}}_{i-1}^A$. Thus, if Theorem 18 is true for a handle at link i, it is also true for a handle at link $i-1$. By induction, the theorem is true for a handle at any link. □

4.4.3 Forward dynamics algorithm

The proof of Theorem 18 gives an algorithm for computing forward dynamics of a serial linkage (Problem 6). The algorithm comprises four main steps:

1. Iterating from the base to the tip of the linkage, compute the velocities of all links (algorithm `compSerialLinkVelocities`, Figure 4.4).

2. Initialize each link's articulated inertia and articulated z.a. force to their isolated counterparts. Also, compute the Coriolis vector for each link (algorithm `initSerialLinks`, Figure 4.7).

3. Iterating from the tip to the base of the linkage, compute the articulated inertia and articulated z.a. forces of each link (algorithm `serialFwdDynamics`, Figure 4.8).

4. Iterating from the base to the tip of the linkage, compute the acceleration of each joint, and the spatial acceleration of each link (algorithm `serialFwdDynamics`, Figure 4.8).

The common subexpressions $\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i$ and $Q_i - \hat{\mathbf{s}}_i'\left(\hat{\mathbf{Z}}_i^A - \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i\right)$ should only be computed once per call.

4.5 Extension to tree-like linkages

The previous sections solve the forward dynamics of serial linkages, such as the one in Figure 4.1. The results can be extended to encompass a more general class of kinematic structures: *tree-shaped* linkages. Consider the complex pendulum shown on the left side of Figure 4.9. This is not a serial linkage, but it does have a tree-like topology. Attached

```
initSerialLinks
```

for $i = 1$ to n

$$\hat{\mathbf{Z}}_i^A \leftarrow \begin{bmatrix} -m_i \mathbf{g} \\ \boldsymbol{\omega}_i \times \mathbf{I}_i \boldsymbol{\omega}_i \end{bmatrix}$$

$$\hat{\mathbf{I}}_i^A \leftarrow \begin{bmatrix} 0 & M_i \\ \mathbf{I}_i & 0 \end{bmatrix}$$

if joint i is prismatic $\hat{\mathbf{c}}_i \leftarrow \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{r}_i) + 2\boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i \end{bmatrix}$

else $\hat{\mathbf{c}}_i \leftarrow \begin{bmatrix} \boldsymbol{\omega}_{i-1} \times \boldsymbol{\nu}_i \\ \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{r}_i) + 2\boldsymbol{\omega}_{i-1} \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) + \boldsymbol{\nu}_i \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) \end{bmatrix}$

Figure 4.7: `initSerialLinks`. Initialize the articulated spatial inertias and articulated z.a. forces to their isolated counterparts, for all links in a serial linkage. Also compute the spatial Coriolis vector for each link.

to each link i is a unique link h that is closer to the fixed base link. Letting h be the parent of i, one can construct a tree representing the linkage, as shown on the right side of Figure 4.9. Each link becomes a node of the tree, and each joint becomes an edge. The links are numbered using the depth-first recursion shown in Figure 4.10

Definition 14 (Tree linkage indices) *Let the links of a tree linkage be indexed using the algorithm of Figure 4.10, and assign each joint the index of the unique link outboard to it. Index the inertial frame as link* 0. *Then the following hold*

1. *The n moving links of the structure and the n joints are numbered $1 \dots n$.*

2. *For any joint i, the link outboard to the joint (the child link) is indexed i, and the link inboard to the joint (the parent link) is indexed h, with $h < i$.*

The forward dynamic algorithm for serial linkages was studied in terms of links. Iterations were performed by visiting each link in tip to base or base to tip order. The algorithm can also be viewed as enumerating over joints, since in a serial linkage there is exactly one joint between every two links, and vice-versa. In studying tree linkages, there

```
serialFwdDynamics

    call compSerialLinkVelocities /* compute all link velocities */
    call initSerialLinks /* initialize Î_i^A, Ẑ_i^A, and ĉ_i for all links */
```

for $i = n$ downto 2

$$\hat{\mathbf{I}}_{i-1}^A \leftarrow \hat{\mathbf{I}}_{i-1}^A + {}_{i-1}\hat{\mathbf{X}}_i \left[\hat{\mathbf{I}}_i^A - \frac{\hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i \hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A}{\hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i} \right] {}_i\hat{\mathbf{X}}_{i-1}$$

$$\hat{\mathbf{Z}}_{i-1}^A \leftarrow \hat{\mathbf{Z}}_{i-1}^A + {}_{i-1}\hat{\mathbf{X}}_i \left[\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A \hat{\mathbf{c}}_i + \frac{\hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i [Q_i - \hat{\mathbf{s}}_i' (\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A \hat{\mathbf{c}}_i)]}{\hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i} \right]$$

$\hat{\mathbf{a}}_0 \leftarrow \hat{\mathbf{0}}$

for $i = 1$ to n

$$\ddot{q}_i = \frac{Q_i - \hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A {}_i\hat{\mathbf{X}}_{i-1} \hat{\mathbf{a}}_{i-1} - \hat{\mathbf{s}}_i' (\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A \hat{\mathbf{c}}_i)}{\hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i}$$

$$\hat{\mathbf{a}}_i = {}_i\hat{\mathbf{X}}_{i-1} \hat{\mathbf{a}}_{i-1} + \hat{\mathbf{c}}_i + \ddot{q}_i \hat{\mathbf{s}}_i$$

Figure 4.8: `serialFwdDynamics`. Compute the accelerations of joints $1, \ldots, n$ of a serial linkage. Input data are the joint positions q_i, joint velocities $\dot{q}_i$, and joint actuator forces/torques Q_i; output data are the joint accelerations $\ddot{q}_i$ ($i = 1, \ldots, n$).

is an advantage to the joint-centric viewpoint. For these, there is not always a unique choice for the next link in a base-to-tip recursion, nor is there a unique starting link for a tip to base recursion. But every joint still has exactly one inboard link and exactly one outboard link. By describing the algorithms in terms of computations to be done at each joint, and visiting the joints in a particular order, the serial linkage algorithm generalizes to tree linkages. Using the joint indexing scheme described above, the algorithm visits the joints in increasing or decreasing order. Pointers to the unique inboard and outboard links to each joint should be stored with the joint.

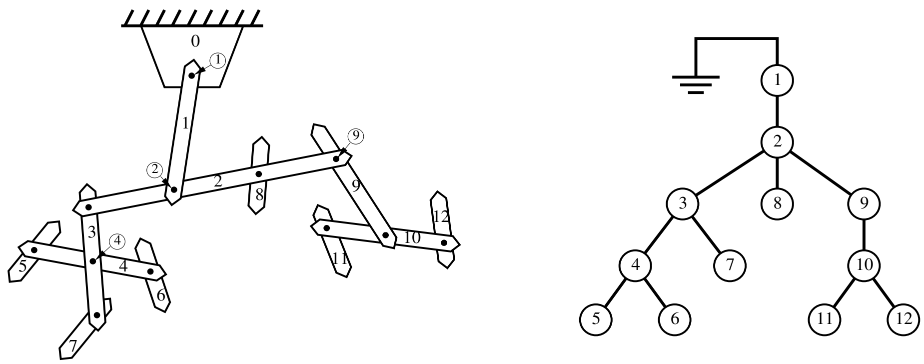

Figure 4.9: Left: *A complex pendulum with a tree-like topology. The moving links are numbered 1 through 12, as are the joints. Four of the joint indices are indicated with circled numbers.* Right: *The corresponding tree representing the kinematic structure.*

```
int numberLinks(tree, idx)


    r ← root link of tree
    r.idx ← idx
    idx ← idx + 1
    for each child c of r
        idx = numberLinks(c, idx)
    return idx
```

Figure 4.10: `int numberLinks(tree, idx)`. Recursively number the links of a kinematic tree, so that every link has higher index than its parent. The routine returns the integer one greater than the highest index in the tree. The routine should initially be called with the entire original tree and `idx = 1`.

4.5.1 Velocity and acceleration propagation through trees

The first step of the forward dynamics algorithm is to compute the absolute velocities of all the links in the structure. This was done in Section 4.2 for serial linkages. Let h be the index of the parent of link i. A little reflection on Figure 4.9 reveals that the absolute velocity of link i may be determined from the absolute velocity of its parent link h, and the velocity of joint i. Thus, the velocity propagation equations (4.12)–(4.13) and (4.16)–(4.17) still hold, with the subscript $i-1$ replaced by h.

The order in which the velocities are propagated through links must be chosen so that the velocity of a link is always computed before it is needed. This is accomplished by iterating over the joints in increasing order, and at each joint propagating the velocity from its inboard link to its outboard link. The algorithm is shown in Figure 4.11. The same quantities may be precomputed as in the serial linkage algorithm.

`compTreeLinkVelocities`

$\boldsymbol{\omega}_0, \mathbf{v}_0, \boldsymbol{\alpha}_0, \mathbf{a}_0 \leftarrow \mathbf{0}$

`for` $i = 1$ `to` n
- $h \leftarrow$ `index of link inboard to joint` i
- `/*` $h =$ `index of parent link,` $i =$ `index of joint & child link */`
- $\mathbf{R} \leftarrow$ `rotation matrix from` $\mathcal{F}_h$ `to` $\mathcal{F}_i$
- $\mathbf{r} \leftarrow$ `radius vector from` $\mathcal{F}_h$ `to` $\mathcal{F}_i$ `(in` $\mathcal{F}_i$ `coordinates)`
- $\boldsymbol{\omega}_i \leftarrow \mathbf{R}\boldsymbol{\omega}_h$
- $\mathbf{v}_i \leftarrow \mathbf{R}\mathbf{v}_h \;+\; \boldsymbol{\omega}_i \times \mathbf{r}$
- `if joint` i `is prismatic`
  - $\mathbf{v}_i \leftarrow \mathbf{v}_i + \dot{q}_i \mathbf{u}_i$
- `else /* joint` i `is revolute */`
  - $\boldsymbol{\omega}_i \leftarrow \boldsymbol{\omega}_i + \dot{q}_i \mathbf{u}_i$
  - $\mathbf{v}_i \leftarrow \mathbf{v}_i + \dot{q}_i(\mathbf{u}_i \times \mathbf{d}_i)$

Figure 4.11: `compTreeLinkVelocities`. Compute the absolute linear and angular velocities of all links of a tree linkage, in link coordinates. Input data are the dynamic state $(\mathbf{q}, \dot{\mathbf{q}})$. Output data are the velocities $\boldsymbol{\omega}_i$ and $\mathbf{v}_i$ for each link.

Propagating accelerations through tree linkages is also analogous to the case for serial linkages. If link h is the parent of link i, the acceleration of link i can be determined from the positions, velocities, and accelerations of link h and joint i. The derivation for acceleration propagation is identical to that for the serial linkage case; the end result is [c.f. Equation (4.23)]:

$$\hat{\mathbf{a}}_i = {}_i\hat{\mathbf{X}}_h \hat{\mathbf{a}}_h + \ddot{q}_i \hat{\mathbf{s}}_i + \hat{\mathbf{c}}_i. \tag{4.30}$$

4.5.2 Articulated inertias and z.a. forces for tree linkages

The next step of the forward dynamics algorithm is to compute the articulated inertias and z.a. forces for all links. For serial linkages, this is done using a tip to base recursion. For a tree linkage, articulated inertias and z.a. forces may be computed directly for links that are the leaves of the tree; for an inner link, the quantities are computed by combining the quantities from the link's children.

Definition 15 (Articulated body of a tree linkage) *For a tree linkage, break the tree at joint i, and consider the connected subtree rooted at link i in isolation, detached from the fixed base. This subtree is an* **articulated body** *of the original tree, and link i is the* **handle** *of the articulated body. If link i is a leaf of the tree, the articulated body is trivial, and is simply the the isolated rigid body of link i.*

The central result to be proved is the counterpart to Theorem 18:

Theorem 19 (Tree linkage articulated motion) *Consider the articulated body of a tree linkage that has link h as a handle ($1 \leq h \leq n$). There exists a spatial matrix $\hat{\mathbf{I}}_h^A$, and a spatial vector $\hat{\mathbf{Z}}_h^A$ such that*

$$\hat{\mathbf{f}}_h^I = \hat{\mathbf{I}}_h^A \hat{\mathbf{a}}_h + \hat{\mathbf{Z}}_h^A. \tag{4.31}$$

Furthermore, $\hat{\mathbf{I}}_h^A$ is independent of the joint velocities and accelerations, and $\hat{\mathbf{Z}}_h^A$ is independent of the joint accelerations.

Theorem 19 will be proved by induction on the height of the articulated body (the maximum number of nodes along a simple path from the handle to some leaf of the subtree). For the base case, consider the height to be 1. Then the articulated body is trivial; it is simply an isolated rigid body that is a leaf of the original linkage. There is no difference between this case and the case of the isolated tip link of a serial linkage. Thus the proof of Section 4.4.1 goes through unchanged for this base case.

For the inductive case, consider the more general articulated body handle shown in Figure 4.12. The handle (link h) may have several links outboard to it; these links are

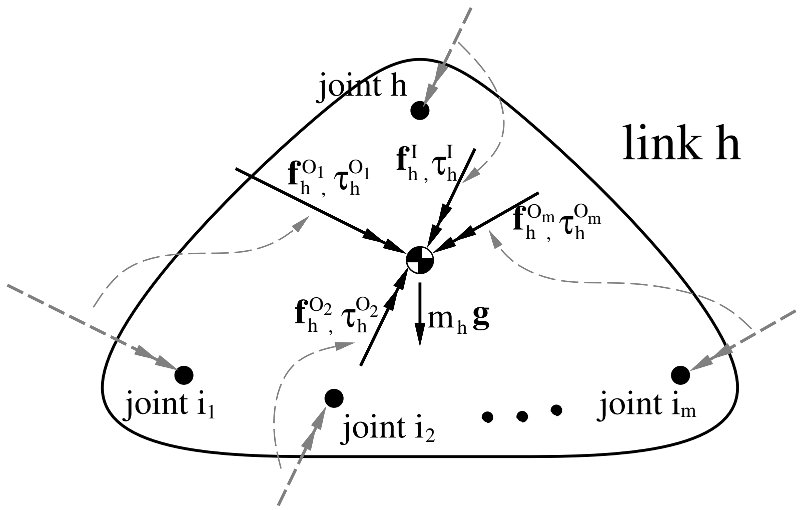

Figure 4.12: *The free body diagram of a non-trivial handle of an articulated body from a tree linkage.*

indexed $i_1, i_2, \ldots, i_m$. The influences applied to link h come from gravity, as well as forces and torques transmitted through the one inboard and m outboard joints. Resolving these influences at the origin of frame $\mathcal{F}_h$, the equations of motion of link h are given by the Newton-Euler equations:

$$\begin{bmatrix} \mathbf{f}_h^I \\ \boldsymbol{\tau}_h^I \end{bmatrix} + \sum_{j=1}^{m} \begin{bmatrix} \mathbf{f}_h^{O_j} \\ \boldsymbol{\tau}_h^{O_j} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{M}_h \\ \mathbf{I}_h & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_h \\ \mathbf{a}_h \end{bmatrix} + \begin{bmatrix} -m_h \mathbf{g} \\ \boldsymbol{\omega}_h \times \mathbf{I}_h \boldsymbol{\omega}_h \end{bmatrix},$$

where the superscript O_j denotes forces or torques imparted through the outboard joint i_j. Using spatial notation and rearranging yields

$$\hat{\mathbf{f}}_h^I = \hat{\mathbf{I}}_h \hat{\mathbf{a}}_h + \hat{\mathbf{Z}}_h - \sum_{j=1}^{m} \hat{\mathbf{f}}_h^{O_j}.$$

Since $\hat{\mathbf{f}}_h^{O_j}$ is the reaction force to $\hat{\mathbf{f}}_{i_j}^I$,

$$\hat{\mathbf{f}}_h^{O_j} = -\,{}_h\hat{\mathbf{X}}_{i_j} \hat{\mathbf{f}}_{i_j}^I.$$

Combining the above two equations,

$$\hat{\mathbf{f}}_h^I = \hat{\mathbf{I}}_h \hat{\mathbf{a}}_h + \hat{\mathbf{Z}}_h + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \hat{\mathbf{f}}_{i_j}^I.$$

Each of the child links i_j is the handle of a subtree of smaller height than the subtree with link h as a handle. The inductive hypothesis (4.31) may be invoked for these child links, giving

$$\hat{\mathbf{f}}^I_h = \hat{\mathbf{I}}_h \hat{\mathbf{a}}_h + \hat{\mathbf{Z}}_h + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} (\hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{a}}_{i_j} + \hat{\mathbf{Z}}^A_{i_j}).$$

Using the acceleration propagation equation (4.30) to express $\hat{\mathbf{a}}_{i_j}$ in terms of $\hat{\mathbf{a}}_h$, and rearranging gives

$$\hat{\mathbf{f}}^I_h = \left(\hat{\mathbf{I}}_h + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \hat{\mathbf{I}}^A_{i_j} {}_{i_j}\hat{\mathbf{X}}_h\right) \hat{\mathbf{a}}_h + \hat{\mathbf{Z}}_h + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \left[\hat{\mathbf{Z}}^A_{i_j} + \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{c}}_{i_j} + \left(\hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j}\right) \ddot{q}_{i_j}\right].$$

The derivation of joint accelerations (see Lemma 7 and the subsequent discussion) makes no assumptions about the topology of the linkage; it applies equally well to tree linkages. The main result (4.27) stated in the current notation is:

$$\ddot{q}_{i_j} = \frac{Q_{i_j} - \hat{\mathbf{s}}'_{i_j} \hat{\mathbf{I}}^A_{i_j} {}_{i_j}\hat{\mathbf{X}}_h \hat{\mathbf{a}}_h - \hat{\mathbf{s}}'_{i_j} \left(\hat{\mathbf{Z}}^A_{i_j} + \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{c}}_{i_j}\right)}{\hat{\mathbf{s}}'_{i_j} \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j}}. \tag{4.32}$$

Combining the above two equations and rearranging gives

$$\begin{aligned} \hat{\mathbf{f}}^I_h \;=\; & \left[\hat{\mathbf{I}}_h + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \left(\hat{\mathbf{I}}^A_{i_j} - \frac{\hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j} \hat{\mathbf{s}}'_{i_j} \hat{\mathbf{I}}^A_{i_j}}{\hat{\mathbf{s}}'_{i_j} \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j}}\right) {}_{i_j}\hat{\mathbf{X}}_h\right] \hat{\mathbf{a}}_h \\ & + \hat{\mathbf{Z}}_h \;+\; \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \left[\hat{\mathbf{Z}}^A_{i_j} + \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{c}}_{i_j} + \frac{\hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j} \left[Q_{i_j} - \hat{\mathbf{s}}'_{i_j} \left(\hat{\mathbf{Z}}^A_{i_j} + \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{c}}_{i_j}\right)\right]}{\hat{\mathbf{s}}'_{i_j} \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j}}\right]. \end{aligned}$$

Comparing this to the desired form (4.31),

$$\hat{\mathbf{I}}^A_h \;=\; \hat{\mathbf{I}}_h + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \left(\hat{\mathbf{I}}^A_{i_j} - \frac{\hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j} \hat{\mathbf{s}}'_{i_j} \hat{\mathbf{I}}^A_{i_j}}{\hat{\mathbf{s}}'_{i_j} \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j}}\right) {}_{i_j}\hat{\mathbf{X}}_h \tag{4.33}$$

$$\hat{\mathbf{Z}}^A_h \;=\; \hat{\mathbf{Z}}_h + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \left[\hat{\mathbf{Z}}^A_{i_j} + \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{c}}_{i_j} + \frac{\hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j} \left[Q_{i_j} - \hat{\mathbf{s}}'_{i_j} \left(\hat{\mathbf{Z}}^A_{i_j} + \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{c}}_{i_j}\right)\right]}{\hat{\mathbf{s}}'_{i_j} \hat{\mathbf{I}}^A_{i_j} \hat{\mathbf{s}}_{i_j}}\right]. \tag{4.34}$$

These equations compute the articulated inertia and articulated z.a. forces for link h of the linkage, given those same quantities for its children links $i_1, \ldots, i_m$. If the $\hat{\mathbf{I}}^A_{i_j}$ are independent of joint velocities and accelerations, than so is $\hat{\mathbf{I}}^A_h$. If the $\hat{\mathbf{Z}}^A_{i_j}$ are independent of joint accelerations, then so is $\hat{\mathbf{Z}}^A_h$. By induction on the height of the subtrees of the handles, Theorem 19 is true for a handle at any link. □

4.5.3 Forward dynamics algorithm for tree linkages

At first glance, the recursive equations (4.33–4.34) seem to imply a more complex computation for tree linkages than for serial ones: multiple articulated inertias and z.a. forces of child links must be combined to compute the corresponding quantities for the parent. But in fact, the same number of back propagations are performed: one for each joint; the algorithm is still $O(n)$. The only difference from the serial algorithm is the order in which inertias, forces, and accelerations are propagated. The forward dynamics algorithm for tree linkages is shown in Figure 4.8. It iterates using a joint centric approach. The articulated inertia and z.a. force of a given link h are accumulated over several iterations, as the contributions from all of link h's children are propagated back. Link h's articulated inertia and z.a. force will reach their final values before they are propagated back to h's parent. When propagating accelerations, link h's acceleration is known before the accelerations of its children are computed. These claims follow from the joint indexing scheme described in Definition 14. The algorithm comprises four steps:

1. Iterating over the joints in increasing order of index, compute the spatial velocities of all links (algorithm `compTreeLinkVelocities`, Figure 4.11).

2. Initialize each link's articulated inertia and articulated z.a. force to their isolated counterparts. Also, compute the Coriolis vector for each link (algorithm `initTreeLinks`, Figure 4.13).

3. Iterating over the joints in decreasing order of index, back propagate the articulated inertia and z.a. forces from the link outboard to the joint to the link inboard to the joint (algorithm `treeFwdDynamics`, Figure 4.14).

4. Iterating over the joints in increasing order of index, compute the acceleration of each joint, and the spatial acceleration of the link outboard to the joint (algorithm `treeFwdDynamics`, Figure 4.14).

4.6 Extension to floating linkages

The algorithms discussed thus far assume *grounded* linkages: link 1 is attached to the inertial frame through a one degree-of-freedom joint. A *floating* linkage is one in `initTreeLinks`

```
for i = 1 to n
```

$$\hat{\mathbf{Z}}_i^A \leftarrow \begin{bmatrix} -m_i\mathbf{g} \\ \boldsymbol{\omega}_i \times \mathbf{I}_i\boldsymbol{\omega}_i \end{bmatrix}$$

$$\hat{\mathbf{I}}_i^A \leftarrow \begin{bmatrix} 0 & M_i \\ \mathbf{I}_i & 0 \end{bmatrix}$$

$h \leftarrow$ `index of link inboard to joint` i `/* link` h `is link` i`'s parent */`

`if joint` i `is prismatic` $\hat{\mathbf{c}}_i \leftarrow \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\omega}_h \times (\boldsymbol{\omega}_h \times \mathbf{r}_i) + 2\boldsymbol{\omega}_h \times \boldsymbol{\nu}_i \end{bmatrix}$

`else` $\hat{\mathbf{c}}_i \leftarrow \begin{bmatrix} \boldsymbol{\omega}_h \times \boldsymbol{\nu}_i \\ \boldsymbol{\omega}_h \times (\boldsymbol{\omega}_h \times \mathbf{r}_i) + 2\boldsymbol{\omega}_h \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) + \boldsymbol{\nu}_i \times (\boldsymbol{\nu}_i \times \mathbf{d}_i) \end{bmatrix}$

Figure 4.13: `initTreeLinks`. Initialize the articulated spatial inertias and articulated z.a. forces to their isolated counterparts, for all links in a tree linkage. Also compute the spatial Coriolis vector for each link.

which link 1 is not attached to the inertial frame. An example of such a system is a mechanical bug: jointed legs are attached to a body (link 1), which is free to move around an environment. Dynamics of such systems are easily modeled with only slight changes to the previous algorithms. These changes are discussed below for the tree linkage forward dynamics algorithm, which is more general than the serial linkage one.

A minimal description of the dynamic state of an n-link floating linkage comprises the position, orientation, linear velocity, and angular velocity of the base link, plus the positions and velocities of the joints:

$$(\mathbf{r}_1, \boldsymbol{\theta}_1, \mathbf{q}; \mathbf{v}_1, \boldsymbol{\omega}_1, \dot{\mathbf{q}}).$$

Here, $\mathbf{r}_1$ is the position of the origin of body frame $\mathcal{F}_1$, and $\boldsymbol{\theta}_1$ is some parameterization of the orientation of this frame; both of these are relative to some fixed frame $\mathcal{O}$. The joint position vector $\mathbf{q}$ is now only $(n-1)$-dimensional. The vectors $\mathbf{v}_1$ and $\boldsymbol{\omega}_1$ are the linear and angular velocities of frame $\mathcal{F}_1$, and $\dot{\mathbf{q}}$ is the $(n-1)$-dimensional joint velocity vector. If unit quaternions (see Appendix A.4) are used to parameterize the orientation of the base link, the

```
treeFwdDynamics

    call compTreeLinkVelocities /* compute all link velocities */
    call initTreeLinks /* initialize Î_i^A, Ẑ_i^A, and ĉ_i for all links */

    for i = n downto 2
        h ← index of link inboard to joint i /* & link i is outboard */
        Î_h^A ← Î_h^A + hX̂i [ Î_i^A − (Î_i^A ŝ_i ŝ'_i Î_i^A) / (ŝ'_i Î_i^A ŝ_i) ] iX̂h
        Ẑ_h^A ← Ẑ_h^A + hX̂i [ Ẑ_i^A + Î_i^A ĉ_i + Î_i^A ŝ_i [Q_i − ŝ'_i(Ẑ_i^A + Î_i^A ĉ_i)] / (ŝ'_i Î_i^A ŝ_i) ]

    â_0 ← 0̂
    for i = 1 to n
        h ← index of link inboard to joint i /* & link i is outboard */
        q̈_i = (Q_i − ŝ'_i Î_i^A iX̂h â_h − ŝ'_i(Ẑ_i^A + Î_i^A ĉ_i)) / (ŝ'_i Î_i^A ŝ_i)
        â_i = iX̂h â_h + ĉ_i + q̈_i ŝ_i
```

Figure 4.14: `treeFwdDynamics`. Compute the accelerations of joints $1, \ldots, n$ of a tree linkage. Input data are the joint positions q_i, joint velocities $\dot{q}_i$, and joint actuator forces/torques Q_i; output data are the joint accelerations $\ddot{q}_i$ $(i = 1, \ldots, n)$.

total dynamic state of an n-link floating linkage has $2n+11$ parameters, as opposed to the $2n$ parameters of a grounded linkage. For a grounded linkage, the forward dynamics algorithm uses the dynamic state to compute $\ddot{\mathbf{q}}$, the n-dimensional joint acceleration vector. For a floating linkage, the algorithm must compute $\mathbf{a}_1$ and $\boldsymbol{\alpha}_1$, the linear and angular accelerations of frame $\mathcal{F}_1$, plus $\ddot{\mathbf{q}}$, the $(n-1)$-dimensional joint acceleration vector.

Changes to the velocity propagation algorithm to handle a floating linkage are minimal. The strategy for computing link velocities is to calculate $\mathbf{v}_i$ and $\boldsymbol{\omega}_i$ from $\mathbf{v}_h, \boldsymbol{\omega}_h$, and $\dot{q}_i$, where link h is the parent of link i. For grounded linkages, the inertial frame serves as the parent of link 1, and $\mathbf{v}_0 = \boldsymbol{\omega}_0 = \mathbf{0}$ starts the iteration. For floating linkages, the velocities $\mathbf{v}_1$ and $\boldsymbol{\omega}_1$ are already part of the dynamic state vector, and so do not need to be computed. The iteration simply starts at joint 2, and proceeds to joint n, at which point the velocities of all the other links will be known.

The calculation of the articulated inertias and articulated z.a. forces for the links needs no adjustment for floating linkages. Featherstone's algorithm always considers articulated bodies, which are collections of connected links that are detached from the inertial frame. Note that in the loop which computes articulated inertias and z.a. forces in Figure 4.14, information about joint 1, the connection between link 1 and the inertial frame, is never used.

In the second loop, accelerations are propagated outward toward the leaf links of the tree. For grounded linkages, the algorithm begins by setting $\hat{\mathbf{a}}_0$ to $\hat{\mathbf{0}}$, and then computes: $\ddot{q}_1$, $\hat{\mathbf{a}}_1$, $\ddot{q}_2$, $\hat{\mathbf{a}}_2$, and so on. For a floating linkage, joint 1 doesn't exist, and the iteration must be bootstrapped in a different manner. The solution comes from Equation 4.31 applied to link 1:

$$\hat{\mathbf{f}}_1^I = \hat{\mathbf{I}}_1^A \hat{\mathbf{a}}_1 + \hat{\mathbf{Z}}_1^A.$$

Since link 1 is disconnected from the inertial frame, the spatial force applied through the inboard joint must be $\hat{\mathbf{0}}$. Thus,

$$\hat{\mathbf{a}}_1 = -\left(\hat{\mathbf{I}}_1^A\right)^{-1} \hat{\mathbf{Z}}_1^A.$$

As long as the individual links all have non-singular mass matrices, which is the case for all real objects, $\hat{\mathbf{I}}_1^A$ is invertible [Fea83]. Once the spatial acceleration $\hat{\mathbf{a}}_1$ is determined, the acceleration propagation proceeds exactly as in the grounded case. The second loop continues, beginning with joint 2. The algorithm remains $O(n)$ since the inversion of the 6×6 $\hat{\mathbf{I}}_1^A$ takes constant time.

Chapter 5

Hybrid Simulation

While there is a large range of contact interactions that can be modeled via trains of impulses, there are also limitations of this approach. Constraints are the natural vehicle for describing many types of contact interactions. This chapter introduces *hybrid simulation*, a method of combining the impulse- and constraint-based simulation paradigms. Impulses are used to transmit unilateral contact forces that arise between separate bodies, while constraints are used to model forces which can push or pull, such as those occurring in a joint bearing. Each method is applied to the contact it is most adept at handling. One main contribution of this chapter is a method for computing lower bounds on the times of impact between rigid bodies that are part of linked structures. Unlike the case for isolated rigid bodies, the initial bounds are sometimes violated, however these violations can always be detected. The collision detection algorithms of Chapter 2 can thus be extended to handle linked bodies. A second major contribution is a differential approach for computing collision impulses arising between rigid bodies that are part of linked structures. This extension of the results presented in Chapter 3 depends on computing a 3×3 multibody collision matrix that encapsulates the collision dynamics at the contact point. An algorithm for computing this matrix is given that runs in $O(n + m)$ time, where n and m are the number of links in the two colliding linked structures. Finally, the chapter describes how a multilevel control architecture can be easily added to an impulse-based simulator, using the one in *Impulse* as an example.

5.1 A spectrum of physical systems

Consider a simple hinge joint (Figure 5.1). In principle, the joint could be

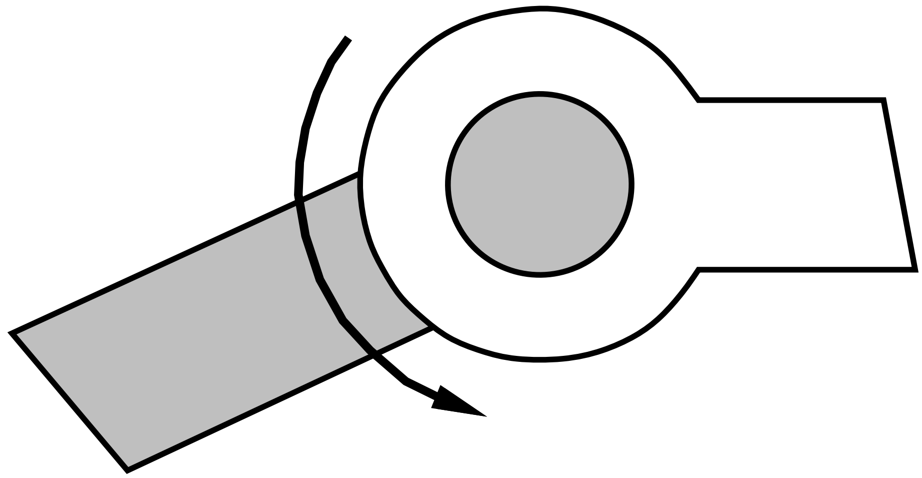

Figure 5.1: *In practice, it is not feasible to model the contact interactions at a hinge joint using an impulse-based approach.*

modeled in an impulse-based way, enforcing the hinge constraint through collisions between the hinge pin and sheath. Due to the enormous amount of collision processing that would be necessary, however, impulse-based simulation would be far too slow. Constraint-based methods should be used instead.

Figure 5.2 depicts a spectrum of physical systems. On the right end are jointed manipulators, a class of systems for which constraint-based simulation approaches are clearly appropriate. The motion of these systems of rigid links is governed by bilateral (equality) constraints imposed at the joints. Techniques to handle bilateral constraints are more developed and robust than those for unilateral (inequality) constraints. The joint constraints are also permanent, facilitating efficient constraint-based simulation. Computing the forward dynamics of these systems is a classical problem of robotics, and a variety of methods exist for doing so. The interaction between the manipulator and the environment (typically occurring at the end effector) is not so clearly a constraint-based interaction; it is often transient and only expressible through unilateral constraints. Over limited intervals of motion, however, the contacts can be modeled through bilateral constraints, and closed-loop methods exist for computing the resulting manipulator dynamics [Lil93].

Slightly to the left on the spectrum lie mechanisms, a class of systems which

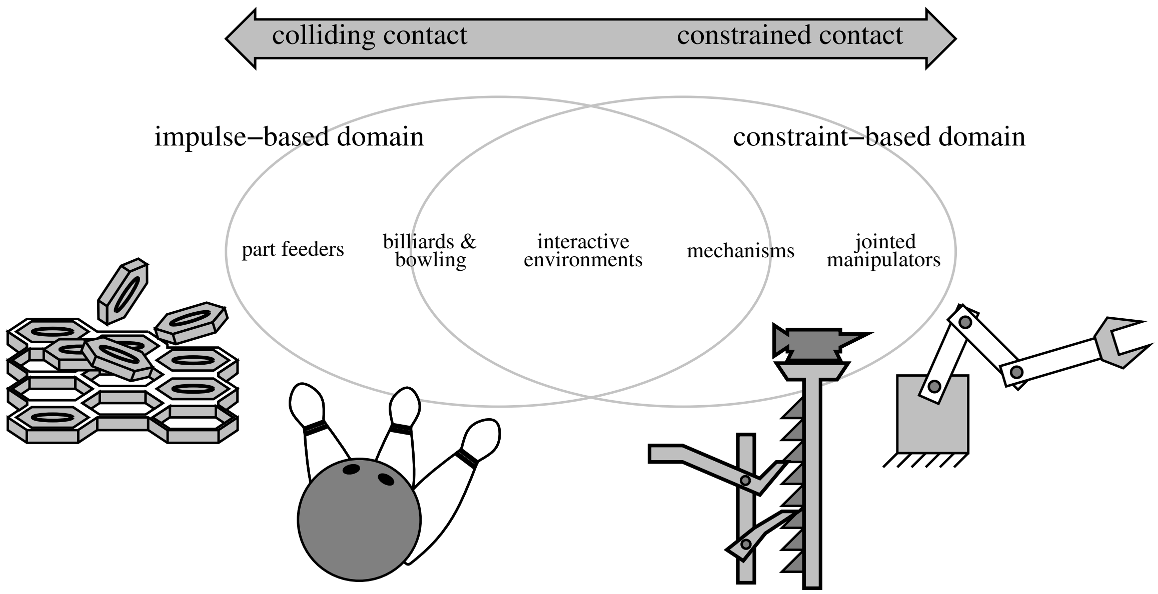

Figure 5.2: *A spectrum of physical systems.*

are also highly constrained. Unilateral constraints are more common in this case. The constraints are also more transient; topologically distinct contact configurations may occur during a complete cycle of the mechanism. Nonetheless, there is still a fair amount of contact coherence, and constraint-based approaches can therefore work well. Baraff's *blockfeeder* and *double-action jack* are good examples of constrained-based simulation of 2D mechanisms [Bar94].

On the other end of the spectrum are highly unconstrained, collision intensive systems like part feeders. In these types of systems, the almost complete lack of contact coherence makes constraint-based methods difficult to apply. Modeling frictional collisions accurately is crucial to correct simulation. For these applications, impulse-based simulation is the natural choice. Slightly to the right on the spectrum lie systems like a bowling alley or a billiards table. In these systems, collisions still occur frequently, and accurate modeling of them is important. Some prolonged contact does occur, but contact coherence is still fairly low, particularly in critical intervals of the simulation, such as when the bowling ball strikes the pins. Impulse-based simulation can adeptly handle these types of systems, including the rolling and sliding contacts that occur.

From a dynamic simulation standpoint, the middle ground of the spectrum has been the least studied. This is probably because no existing simulation paradigms are a natural fit for the interesting applications that lie there. Consider simulating a user-controlled agent within an interactive virtual environment. A simulator must respond to erratic, unpredictable behavior on the part of the user-controlled agent or other autonomous agents in the environment. Collisions provide much of the interaction between agents and their environment. The agents may bump into each other or other objects, and may move or throw objects around. Contact modes are transient as agents grab, push, or pull objects. Settling phases, as objects come to rest after being positioned or dropped by agents, involve more transient contact. There are also many types of contact best modeled with constraints. Hinged doors between rooms, simple mechanical structures like levers, and the joints of the agents all entail permanent, bilateral constraints. There may be stacks of objects, or other situations where contact coherence is high and constrained-based methods can effectively model the non-penetration between bodies.

An interactive virtual environment is just one example of an application that seems to require both impulse-based and constraint-based techniques for efficient simulation; ways are needed to blend the approaches. This chapter discusses *hybrid simulation*, which is a first step in this direction. Hybrid simulation extends the pure impulse-based methods discussed thus far to accommodate permanent, bilateral constraints. Thus systems of rigid bodies connected by joints may be handled. Adjustments to the collision detection scheme are required. Also, the calculation of collision impulses must account for the dynamics of the bodies that are linked to the colliding bodies. A connection to the simple rigid body collision response algorithm is made through the collision matrix $\mathbf{K}$; much of the the theory discussed in Chapter 3 then carries over to the case of linked bodies. Finally, once the impulses are computed, they must be propagated through the linked systems of bodies, instantaneously changing the velocities of all bodies in the linkage.

Collections of linked rigid bodies are called multibodies. In this chapter, multibodies are restricted to the following definition:

Definition 16 *A* **multibody system***, or multibody, is a collection of rigid bodies linked together through one degree of freedom revolute or prismatic joints, with tree-like topology. That is, the system is a single connected piece with no loops among the links. One link is specified as the* base link*, and this link may be connected to an inertial frame via a revolute or prismatic joint, or via a full six degree of freedom joint, resulting in a* floating *multibody.*

The restrictions imposed by this definition are not essential, but facilitate use of the forward dynamics algorithms discussed in the last chapter. There are extensions of Featherstone's algorithm for more general joints [Lil93, JR]. There are also methods of computing linkage dynamics in the presence of kinematic loops [Lil93, LNPE92, dJB94, Bar96]. Multibody impulses could still be computed in cases of these more general systems; the construction of the collision matrix $\mathbf{K}$ would involve using more the general forward dynamics algorithms instead of those presented in the last chapter.

5.2 Collision detection

For collision detection purposes, a multibody is broken into its component rigid bodies. When a body passes in the vicinity of a multibody, the narrow phase collision detection system will track closest points between the body and the nearby multibody links. Links that are not close to the body will be discarded from consideration by the broad phase detection. The collision detection system can be extended to handle multibodies using the two interface routines described in Section 2.5: a swept volume routine and a TOI coefficients routine.

The motion of a rigid body that is part of a multibody is much more complex that that of a rigid body or scripted body. Conservative bounds on swept volumes and TOIs are difficult to derive, and the requirement that these bounds be conservative is abandoned for multibodies. Instead, *Impulse* makes aggressive predictions that are hopefully valid bounds in the "average" case. If an aggressive bound is violated, this violation must be detected, and the simulator must recover gracefully.

5.2.1 Constrained body swept volumes

Recall the goal of the swept volume routine: Given the state of a body at the current time t_0, and a time interval Δt, the routine returns an axes-aligned bounding box which encloses the body's center of mass during the interval $[t_0, t_0 + \Delta t]$. For bodies moving ballistically or along simple scripted trajectories, this can be done. For bodies which are part of a multibody, it is very difficult. But one can assume the coordinates of the center of mass $\mathbf{r}$ will follow second-order trajectories:

$$
\begin{aligned}
r_x(t) &= r_x(t_0) + \dot{r}_x(t_0)(t - t_0) + \frac{1}{2}\ddot{r}_x(t_0)(t - t_0)^2 \\
r_y(t) &= r_y(t_0) + \dot{r}_y(t_0)(t - t_0) + \frac{1}{2}\ddot{r}_y(t_0)(t - t_0)^2
\end{aligned}
$$

$$r_z(t) \quad = \quad r_z(t_0) + \dot{r}_z(t_0)(t - t_0) + \frac{1}{2}\ddot{r}_z(t_0)(t - t_0)^2.$$

The quantities $\mathbf{r}(t_0)$ and $\dot{\mathbf{r}}(t_0)$ are inferred from the current dynamic state of the multibody, and $\ddot{\mathbf{r}}(t_0)$ is a by-product of the forward dynamics algorithm: it is just the linear acceleration $\mathbf{a}$ of the link. These accelerations can be stored so that their current values are always available.

Using these second-order approximations make it easy to predict the minimum and maximum r_x, r_y, and r_z values over the interval $[t_0, t_0 + \Delta t]$. It makes sense to pad these values somewhat, because of the higher order terms which are neglected in the above equations; this means returning a slightly larger bounding box than that predicted by the model. The tradeoff is that as box size increases, so does the chance that more body pairs will be promoted to the active collision heap, perhaps unnecessarily. The padding scheme used in *Impulse* is to scale $\dot{\mathbf{r}}(t_0)$ and $\ddot{\mathbf{r}}(t_0)$ by a factor 1.1, before using them in the above equations.

The position and velocity of a multibody is numerically integrated, using a derivative evaluation routine such as `treeFwdDynamics` (Chapter 4). Each time the derivative evaluation routine is called, the positions of the links are determined. The minimum and maximum coordinate values of each link's center of mass are tracked over the entire integration interval; at the end of the integration, it is easy to check if these coordinates remained within the bounding box predicted by the swept volume routine.

When a violation does occur, the correct bounding box for the interval is known, since the minimum and maximum center of mass coordinate values are known. The bounding box in the spatial hash table is updated to reflect this new information. If the new box intersects no other boxes than the old box did, then no body pairs should be added to the collision heap, and the integration is still valid. If new possible collision pairs are discovered, then the times to impact between these new pairs are computed, and the pairs are added to the collision heap. If this does not cause the body pair at the top of the heap to change, the integration is still valid. Otherwise, a new collision check, which was previously missed because of the incorrect bounding box, has been discovered, and this collision check should have been performed *earlier* than the ending time of the last integration. In this case, the simulation must be backed up. The bodies' states are restored to their values at the beginning of the last integration, and the integration is redone, this time stopping in time for the earlier collision check. After this second integration, there will be no bounding box violations, since all of the invalid *a priori* boxes have been replaced by the valid *a posteri* ones. Figure 5.3 details the entire process.

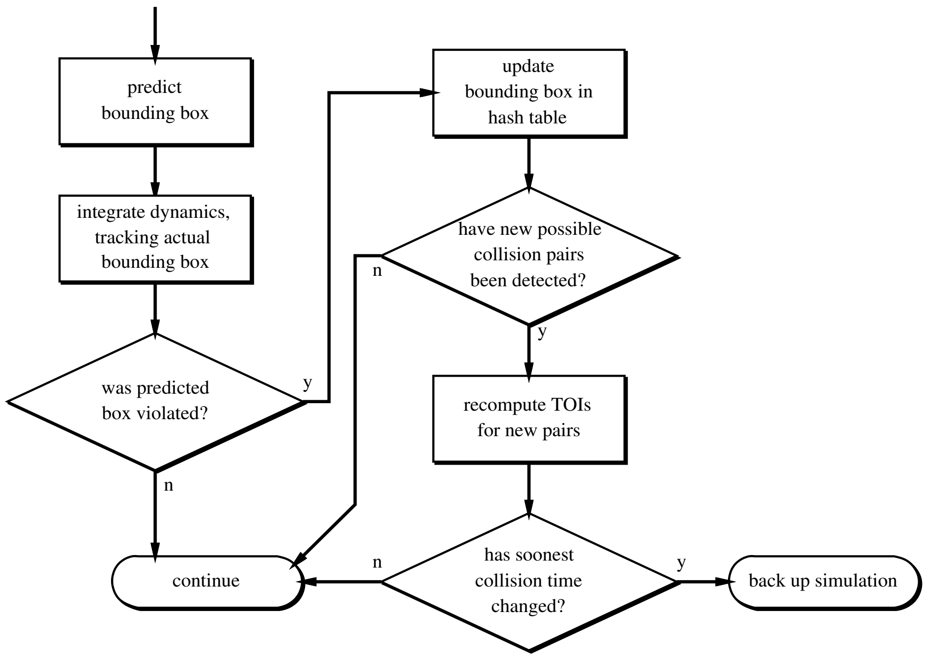

Figure 5.3: *The checking performed to detect aggressive bounding box violations, and to determine if the simulation should be backed up as a result of the violation.*

5.2.2 Constrained body TOI coefficients

The narrow phase collision detection system requires a time of impact coefficient routine: given the dynamic state of a body at the current time t_0, and directional vector $\hat{\mathbf{d}}$, return coefficients A and B such that the distance that any point on the body travels along the direction $\hat{\mathbf{d}}$ is bounded above by

$$A(t - t_0)^2 + B(t - t_0).$$

Conservative values for A and B can be calculated for ballistic bodies and certain scripted bodies; for constrained bodies, this remains an open problem. The approach taken in *Impulse* is to abandon the requirement that the predictions be conservative, and attempt to compute reasonable coefficients which are usually valid. A second order model of motion is again used. The coefficients computed for constrained bodies are:

$$
\begin{aligned}
A &= \frac{1}{2}\mathbf{a}(t_0)\cdot\hat{\mathbf{d}} \\
B &= \mathbf{v}(t_0)\cdot\hat{\mathbf{d}} + r_{\max}\|\boldsymbol{\omega}(t_0)\|
\end{aligned}
$$

These are very similar to the coefficients for ballistic rigid bodies, (2.2)–(2.3). One difference is that the gravity vector $\mathbf{g}$ is replaced with the current acceleration of the body, $\mathbf{a}(t_0)$, a more general acceleration which also includes the effects of gravity. The $\omega_{\max}$ term is replaced with $\|\boldsymbol{\omega}(t_0)\|$, the current angular velocity, since it is difficult to compute a tight bound the angular velocity of a multibody link over some future time interval. Because of the approximations made in this model, the coefficients are padded by a safety factor. In *Impulse* A and B are computed as above, and then increased by 0.2 times their respective absolute value. This reduces the chance that the coefficients will be violated.

The strategy for detecting time of impact violations is similar to that for detecting bounding box violations: the discrete data provided during numerical integration is tracked. TOI predictions are monitored using *slack structures*. For each TOI estimate made for a constrained body, there is a slack structure that tracks the progress of points on this body in the direction $\hat{\mathbf{d}}$. The components of a slack structure are

1. The unit vector $\hat{\mathbf{d}}$, along whose direction motion of points of the body is monitored.

2. The constants A and B, which bound the distance traveled by points on the body in the direction $\hat{\mathbf{d}}$.

3. The base time t_b at which the slack was initialized.

4. An upper bound δ on the distance traveled by any point on the body in the direction $\hat{\mathbf{d}}$, since time t_b. This is a truly conservative bound, based on values tracked during dynamic integration.

5. The time t_l at which the slack bound δ as last updated.

When a TOI prediction involving a constrained body B is made, a slack is initialized. The coefficients A and B are computed as described above, the base time t_b and last time t_l are set to the current time, and the slack bound δ is set to 0. During integration, the maximum and minimum values of each component of the body's linear and angular velocity are tracked. After the integration, maximum distance any point on the body could have traveled along $\hat{\mathbf{d}}$ is computed, using the tracked integration values; this distance is added to δ. A check is made to insure

$$\delta \leq A(t_0 - t_b)^2 + B(t_0 - t_b),$$

where t_0 is the current time. If this check fails, a TOI violation has occurred. The algorithm to update the slack is shown in Figure 5.4.

```
updateSlack

    dt ← t_0 − t_l
    if (dt < 0) return

    v_min ← minimum components of linear velocity over last integration
    v_max ← maximum components of linear velocity over last integration
    ω_max ← maximum magnitude components of angular velocity
        over last integration

    k ← 0
    if (d̂_x < 0) k ← k + v_min_x d̂_x ; else k ← k + v_max_x d̂_x ;
    if (d̂_y < 0) k ← k + v_min_y d̂_y ; else k ← k + v_max_y d̂_y ;
    if (d̂_z < 0) k ← k + v_min_z d̂_z ; else k ← k + v_max_z d̂_z ;
    k ← k + r_max * (ω_max_x sqrt(1 − d̂_x^2) + ω_max_y sqrt(1 − d̂_y^2) + ω_max_z sqrt(1 − d̂_z^2))
    δ ← δ + k dt
    if (δ > A(t_0 − t_b)^2 + B(t_0 − t_b)) signal violation
    else t_l ← t_0
```

Figure 5.4: `updateSlack`. Update a slack structure and signal a TOI violation if necessary. The variables t_l, δ, A, B, and t_b are all fields of the slack structure. The variable t_0 is the current simulation time.

When a violation is detected, the offending TOI is recomputed, using the now known values for its motion during the integration step. The simulation is backed up to the state at the beginning of the integration interval in which the violation occurred, and the integration is redone. In this way, the collision detection system avoids missing collisions do to overly aggressive TOI coefficients.

5.3 Collision response

The algorithms developed in Chapter 4 address the dynamics of multibodies in isolation, but say nothing about their dynamics when they come into contact with other bodies. Contact forces can be added to the model if one has a way to compute them. Impulse-based simulation adopts an alternative approach: all contacts are modeled through trains of collision impulses. Chapter 3 described how to compute these impulses for the case of rigid body collisions; this section handles the case of multibody collisions.

Problem 8 (Multibody collision response) *Given two colliding rigid bodies, each of which may be part of a larger multibody, compute the impulse pair to be applied at the two contact points, and the instantaneous changes in velocity of the colliding bodies, as well as any changes in the velocity of other links of the multibodies.*

The response of a multibody to an impulse is simpler than its response to an applied force. Consider applying some force to a link of a multibody. The joint accelerations depend on this force, but because of other forces due to gravity and possibly joint actuators, and because of the velocity dependent Coriolis and centripetal forces, the joint accelerations are not linear functions of the applied force. On the other hand, when a collision impulse is applied to the multibody, it dominates all other (ordinary magnitude) forces that are acting on the multibody, including the external forces and the velocity dependent forces. Unlike impulses, ordinary magnitude forces have no effect on dynamics over an infinitesimal interval. As a result, all of the nonlinear terms drop out of the dynamic equations during the interval of collision, resulting in a linear relationship between the applied impulse and the instantaneous changes in joint velocities. This implies a linear relationship between applied impulse and change in contact point velocity, and the linearity can be exploited just as it was for rigid body collision response.

For physical realism, friction and non-elastic behavior at the contact point should be included in the model, just as they were for rigid body collisions. Much of physical derivation behind rigid body collision response can be applied directly to the multibody case. The key is computing the multibody collision matrix, the counterpart to the rigid body collision matrix.

5.3.1 Collision response: a robotics perspective

The existence of the multibody collision matrix can be shown using the classical robotics formulation of multibody dynamics. Consider a robot in contact with its environment, as in Figure 5.5. The equations of motion for the robot in this situation are

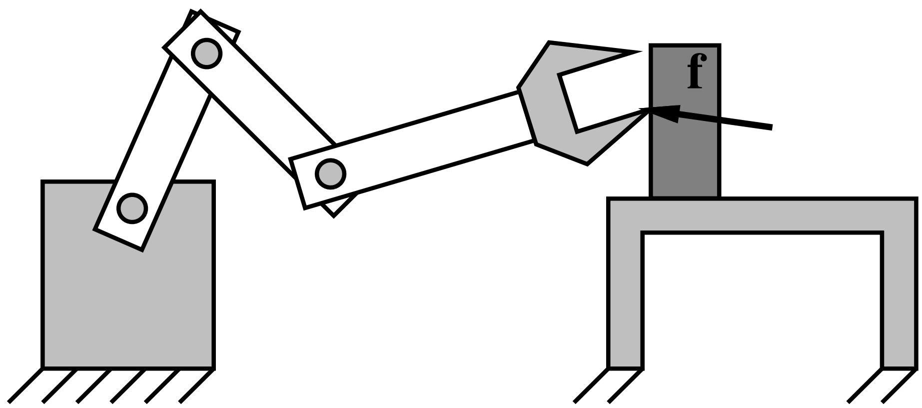

Figure 5.5: *A robot in contact with its environment.*

[Lil93]:

$$\ddot{\mathbf{q}}(t) = \mathbf{H}^{-1}(\mathbf{q}(t)) \left[\mathbf{Q}(t) - \mathbf{C}(\mathbf{q}(t), \dot{\mathbf{q}}(t))\dot{\mathbf{q}(t)} - \mathbf{G}(\mathbf{q}(t)) + \mathbf{J}^T(\mathbf{q}(t))\ \hat{\mathbf{f}}(t)\right]. \tag{5.1}$$

Here, $\mathbf{q}$ is the vector of joint positions; $\mathbf{H}$ is the joint-space inertia matrix; $\mathbf{C}$ is the Coriolis matrix, and $\mathbf{C}\dot{\mathbf{q}}$ contains all of Coriolis and centripetal acceleration terms; $\mathbf{G}$ describes the effects of external forces like gravity; $\mathbf{J}$ is the $6 \times n$ Jacobian matrix at the end effector; $\hat{\mathbf{f}}$ is the external force and torque applied to the end effector (the caret is a reminder that this is a 6×1 spatial force); and $\mathbf{Q}$ is a vector of the magnitudes of forces and torques applied by the joint actuators. All quantities on the right-hand side are known or computable from the dynamic state, so this equation can be directly integrated to determine the forward dynamics of the robot. This equation also shows that the joint accelerations are not a linear function of the force $\hat{\mathbf{f}}$ applied to the end effector.

Suppose in Figure 5.5 that the end effector is *colliding* with the block. The collision produces a very large external force $\hat{\mathbf{f}}$ acting on the multibody for a short period of time t_c.

For rigid bodies, the collision time is infinitesimal; consider the limit as $t_c \to 0$. The change in joint velocities during the collision is

$$\Delta\dot{\mathbf{q}} = \lim_{t_c \to 0} \int_0^{t_c} \ddot{\mathbf{q}}(t)\ dt.$$

In the limit, the collision force grows arbitrarily large, and the terms $\mathbf{Q}(t), \mathbf{C}(\mathbf{q}(t), \dot{\mathbf{q}}(t))$, and $\mathbf{G}(\mathbf{q}(t))$ become negligible. The same assumption is made in rigid body collisions, when finite magnitude forces like gravity are ignored over the duration of a collision. Substituting (5.1) into the above integral, and dropping negligible terms,

$$\Delta\dot{\mathbf{q}} = \lim_{t_c \to 0} \int_0^{t_c} \mathbf{H}^{-1}(\mathbf{q}(t))\ \mathbf{J}^T(\mathbf{q}(t))\ \hat{\mathbf{f}}(t)\ dt.$$

In the limit, $\mathbf{q}(t)$ is a constant, just as the positions of the bodies are constant during isolated rigid body collisions. Pulling constants outside the integral,

$$\Delta\dot{\mathbf{q}} = \mathbf{H}^{-1}(\mathbf{q})\ \mathbf{J}^T(\mathbf{q})\ \lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{f}}(t)\ dt = \mathbf{H}^{-1}(\mathbf{q})\ \mathbf{J}^T(\mathbf{q})\ \hat{\mathbf{p}},$$

where $\hat{\mathbf{p}}$ is the 6×1 spatial collision impulse.

The change in the end effector velocity is related to the change in joint velocities through the Jacobian:

$$\begin{bmatrix} \Delta\boldsymbol{\omega} \\ \Delta\mathbf{v} \end{bmatrix} = \mathbf{J}(\mathbf{q})\ \Delta\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})\ \mathbf{H}^{-1}(\mathbf{q})\ \mathbf{J}^T(\mathbf{q})\ \hat{\mathbf{p}}.$$

The 6×6 matrix $\mathbf{J}(\mathbf{q})\ \mathbf{H}^{-1}(\mathbf{q})\ \mathbf{J}^T(\mathbf{q})$ on the right-hand side is the inverse of the operational space inertia matrix $\boldsymbol{\Lambda}$ developed by Khatib [Kha87]. This matrix describes the apparent mass of the end effector to an observer exerting forces and torques on it.

The robot is one of two bodies involved in the collision; on the robot, call the velocity at the contact point $\mathbf{u}_1$. The change in $\mathbf{u}_1$ is related to the change in the end effector's velocities by

$$\Delta\mathbf{u}_1 = \Delta\mathbf{v} + \Delta\boldsymbol{\omega} \times \mathbf{r},$$

where $\mathbf{r}$ is the offset vector of the collision point in the end effector frame. Letting $\mathbf{1}$ denote the 3×3 identity matrix, this can be written as a matrix equation:

$$\begin{aligned} \Delta\mathbf{u}_1 &= [-\tilde{\mathbf{r}}, \mathbf{1}] \begin{bmatrix} \Delta\boldsymbol{\omega} \\ \Delta\mathbf{v} \end{bmatrix} \\ &= \underbrace{[-\tilde{\mathbf{r}}, \mathbf{1}]\, \mathbf{J}(\mathbf{q})\ \mathbf{H}^{-1}(\mathbf{q})\ \mathbf{J}^T(\mathbf{q})}_{\mathbf{W}}\ \hat{\mathbf{p}}, \end{aligned}$$

The last equation shows the linear relationship between the collision impulse $\hat{\mathbf{p}}$ and the change in contact point velocity $\mathbf{u}_1$; they are related by a 3×6 *constant* matrix $\mathbf{W}$. The impulse $\hat{\mathbf{p}}$ is a spatial quantity, containing both force and torque components. Both components might be useful, for example, when modeling torsional friction during impact. For rigid-body collisions, a simpler model is typically used: only pure forces can be generated at the contact point. For this case, $\hat{\mathbf{p}} = (\mathbf{p}, \mathbf{0})^T$, where $\mathbf{p}$ is a pure force impulse, with only three components. Denoting the left 3×3 sub-matrix of $\mathbf{W}$ by $\mathbf{K}_1$,

$$\Delta \mathbf{u}_1 = \mathbf{K}_1 \ \mathbf{p}.$$

The other body involved in the collision also experiences a collision impulse and a corresponding change in contact point velocity. By Newton's third law, the impulse experienced by the second body must be $-\mathbf{p}$. Calling the contact point velocity on the second body $\mathbf{u}_2$, one finds

$$\Delta \mathbf{u}_2 = \mathbf{K}_2 \ (-\mathbf{p}).$$

If the second colliding body is a link of a multibody, $\mathbf{K}_2$ can be derived by the process described above for $\mathbf{K}_1$; if it is just a rigid body, the formulas from Chapter 3 may be used. Since the relative contact point velocity $\mathbf{u}$ between the two bodies is given by

$$\mathbf{u} = \mathbf{u}_1 - \mathbf{u}_2,$$

The above three equations imply:

$$\Delta \mathbf{u} = \underbrace{(\mathbf{K}_1 + \mathbf{K}_2)}_{\mathbf{K}} \ \mathbf{p}. \tag{5.2}$$

Compare this equation with its counterpart (3.5) derived for simple rigid bodies: they are identical.[1] In both cases, the change in relative contact point velocity is related to the total impulse applied at the contact point through a constant matrix. Once this connection is made, the collision impulse can be found by forming the collision ODE and performing a collision integration, as described in Section 3.2. The same algorithm for computing collision impulses between rigid bodies can be applied to colliding multibodies. The above derivation is valid for multibodies that collide at any of their links, not only the

[1]In the current derivation, the explicit dependence of $\mathbf{p}$ and $\mathbf{u}$ on a collision parameter γ has been suppressed for clarity.

at end effectors. The only change to the above analysis is that the Jacobian in (5.1) must be referenced to the colliding link.

One nice feature of hybrid simulation is that some kinematic loops are handled automatically. Consider again the manipulator of Figure 5.5. When it contacts the environment, it forms a kinematic loop with the environment. With a constraint-based approach, this situation requires special handling: the manipulator dynamics must be combined with the kinematic constraint imposed at the end effector [Lil93]. Under hybrid simulation, the contact force between the end effector and the environment is modeled with collision impulses that are propagated through the manipulator. Once the collision response algorithm is developed, no changes are needed in the manipulator forward dynamics algorithms. This same technique is used to model the dynamics of wheeled vehicles that form kinematic loops with the ground; examples are described in Chapter 7.

5.3.2 Articulated body collision dynamics

The previous section demonstrated the existence of the multibody collision matrix $\mathbf{K}$; an algorithm for computing it is now developed. Only half the problem is considered: computing the 3×3 matrix $\mathbf{K}_i$, which relates the impulse applied to one of the colliding bodies to the change in contact point velocity on that body. This problem is solved once for each body, and $\mathbf{K}_1$ and $\mathbf{K}_2$ are combined to form the final multibody collision matrix $\mathbf{K}$, as in (5.2). One way to compute $\mathbf{K}_i$ has been shown already: it is the left half of the 3×6 matrix

$$\mathbf{W} = [-\tilde{\mathbf{r}}, \mathbf{1}]\, \mathbf{J}(\mathbf{q})\ \mathbf{H}^{-1}(\mathbf{q})\ \mathbf{J}^T(\mathbf{q}).$$

Computing this product explicitly, however, results in an $O(n^3)$ algorithm for an n-link multibody, due to the inversion of $\mathbf{H}(\mathbf{q})$. Since collisions occur so frequently in an impulse-based simulation, computing $\mathbf{K}_i$ efficiently is important. By studying the dynamics of articulated bodies of the full multibody during a collision, an $O(n)$ algorithm for computing $\mathbf{K}_i$ may be derived. The algorithm will be derived for tree-like multibodies; these include serial multibodies as a subclass.

Consider the dynamics of a tree-like articulated body during a collision. From (4.31),

$$\lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{f}}_h^I \, dt = \lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{I}}_h^A \hat{\mathbf{a}}_h \, dt + \lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{Z}}_h^A \, dt.$$

In the limit as t_c approaches zero, the collision force acting on the body is infinite, which gives rise to infinite collision forces acting through joints of the linkage. Call the limit of the integral on the left hand side $\hat{\mathbf{p}}_h^I$, the spatial impulse exerted on link h through the inboard joint. The articulated inertia $\hat{\mathbf{I}}_h^A$ depends on link masses and joint positions, which are constant as $t_c \to 0$, thus

$$\begin{aligned}
\hat{\mathbf{p}}_h^I &= \hat{\mathbf{I}}_h^A \int_0^{t_c} \hat{\mathbf{a}}_h \, dt + \lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{Z}}_h^A \, dt \\
&= \hat{\mathbf{I}}_h^A \Delta \hat{\mathbf{v}}_h + \lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{Z}}_h^A \, dt.
\end{aligned}$$

The quantity $\Delta\hat{\mathbf{v}}_h$ is the total change in spatial velocity of link h during the collision; it is the limit of infinite accelerations integrated over an infinitesimal interval. The integral of z.a. forces contain some terms which vanish in the limit, and others which remain. From (4.34), the last term in the above equation is

$$\lim_{t_c \to 0} \int_0^{t_c} \left\{ \hat{\mathbf{Z}}_h + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \left[\hat{\mathbf{Z}}_{i_j}^A + \hat{\mathbf{I}}_{i_j}^A \hat{\mathbf{c}}_{i_j} + \frac{\hat{\mathbf{I}}_{i_j}^A \hat{\mathbf{s}}_{i_j} \left[Q_{i_j} - \hat{\mathbf{s}}_{i_j}' \left(\hat{\mathbf{Z}}_{i_j}^A + \hat{\mathbf{I}}_{i_j}^A \hat{\mathbf{c}}_{i_j} \right) \right]}{\hat{\mathbf{s}}_{i_j}' \hat{\mathbf{I}}_{i_j}^A \hat{\mathbf{s}}_{i_j}} \right] \right\} dt.$$

Velocities remain finite in magnitude during a collision, therefore so do the spatial Coriolis forces $\mathbf{c}_{i_j}$. The joint actuators can not apply impulses, so the Q_{i_j} are also finite. The quantities ${}_h\hat{\mathbf{X}}_{i_j}$, $\hat{\mathbf{I}}_{i_j}^A$, and $\hat{\mathbf{s}}_{i_j}$ are all constant as $t_c \to 0$. Pulling these constants outside the integrals, and dropping the finite terms that vanish in the limit,

$$\lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{Z}}_h^A \, dt = \lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{Z}}_h \, dt + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \left[\mathbf{1} - \frac{\hat{\mathbf{I}}_{i_j}^A \hat{\mathbf{s}}_{i_j} \hat{\mathbf{s}}_{i_j}'}{\hat{\mathbf{s}}_{i_j}' \hat{\mathbf{I}}_{i_j}^A \hat{\mathbf{s}}_{i_j}} \right] \lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{Z}}_{i_j}^A \, dt, \tag{5.3}$$

where $\mathbf{1}$ is the 6×6 identity matrix. Defining the isolated and articulated zero acceleration (z.a.) impulses as

$$\hat{\mathbf{Y}}_h = \lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{Z}}_h \, dt \tag{5.4}$$

$$\hat{\mathbf{Y}}_h^A = \lim_{t_c \to 0} \int_0^{t_c} \hat{\mathbf{Z}}_h^A \, dt, \tag{5.5}$$

(5.3) becomes:

$$\hat{\mathbf{Y}}_h^A = \hat{\mathbf{Y}}_h + \sum_{j=1}^{m} {}_h\hat{\mathbf{X}}_{i_j} \left[\mathbf{1} - \frac{\hat{\mathbf{I}}_{i_j}^A \hat{\mathbf{s}}_{i_j} \hat{\mathbf{s}}_{i_j}'}{\hat{\mathbf{s}}_{i_j}' \hat{\mathbf{I}}_{i_j}^A \hat{\mathbf{s}}_{i_j}} \right] \hat{\mathbf{Y}}_{i_j}^A. \tag{5.6}$$

Lemma 8 *Consider a tree linkage that experiences a collision impulse at link k. Let $\hat{\mathbf{p}}_{coll}$ be the spatial vector comprising the impulsive collision force and torque, resolved at the center of the body frame of link k. Then the isolated z.a. impulses of the links are given by*

$$\hat{\mathbf{Y}}_i = \begin{cases} \hat{\mathbf{0}}, & i \neq k \\ -\hat{\mathbf{p}}_{coll}, & i = k \end{cases}$$

Proof: By definition, $\hat{\mathbf{Z}}_i$ is the force applied at the inboard joint that cancels out the Coriolis, gravitational, and external forces acting on link i, when link i is considered detached from the rest of the linkage. $\hat{\mathbf{Y}}_i$ is the integral of this force over time as $t_c \to 0$. The force needed to oppose the finite magnitude Coriolis and gravitational forces may be ignored, since they vanish in the limit. If $i \neq k$, than there are no other external forces acting on the link, and so $\hat{\mathbf{Y}}_i = \hat{\mathbf{0}}$. If $i = k$, then $\hat{\mathbf{Z}}_i$ must also cancel the spatial collision force. This force becomes the impulse $\hat{\mathbf{p}}_{\mathrm{coll}}$ in the limit, thus $\hat{\mathbf{Y}}_i = -\hat{\mathbf{p}}_{\mathrm{coll}}$. $\square$

Lemma 9 *Consider a tree linkage that experiences a collision impulse at link k. If link k is not in the subtree rooted at link h, $\hat{\mathbf{Y}}_h^A = \hat{\mathbf{0}}$.*

Proof: The proof is by induction on the height of the subtree rooted at link h. If the height is 1, link h is a leaf link with no descendents. By (5.6), $\hat{\mathbf{Y}}_h^A = \hat{\mathbf{Y}}_h$. Since $h \neq k$, $\hat{\mathbf{Y}}_h = \hat{\mathbf{0}}$ by Lemma 8. Suppose now the height of link h is greater than 1. Index link h's children $i_1, ..., i_m$. Each of the children is the root of a subtree of lesser height which also does not contain link k. By the inductive hypothesis, $\hat{\mathbf{Y}}_{i_j}^A = \hat{\mathbf{0}}$, $1 \leq j \leq m$. From (5.6), $\hat{\mathbf{Y}}_h^A = \hat{\mathbf{Y}}_h = \hat{\mathbf{0}}$. $\square$

Lemma 9 implies that articulated z.a. impulses need only be calculated along the chain from the colliding link k up to the base link; the articulated z.a. impulse for all other links is $\hat{\mathbf{0}}$ (Figure 5.6). Since $\hat{\mathbf{Y}}_{i_j}^A = \hat{\mathbf{0}}$ for all children i_j of the colliding link k, $\hat{\mathbf{Y}}_k^A = \hat{\mathbf{Y}}_k$ by (5.6). By Lemma 8,

$$\hat{\mathbf{Y}}_k^A = -\hat{\mathbf{p}}_{\mathrm{coll}}. \tag{5.7}$$

Now consider link h, a proper ancestor of link k. It has exactly one child link which is a (not necessarily proper) ancestor of link k; any other children are roots of subtrees which do not contain link k. For these other children i_j, $\hat{\mathbf{Y}}_{i_j}^A = \hat{\mathbf{0}}$ by Lemma 9. Thus only one

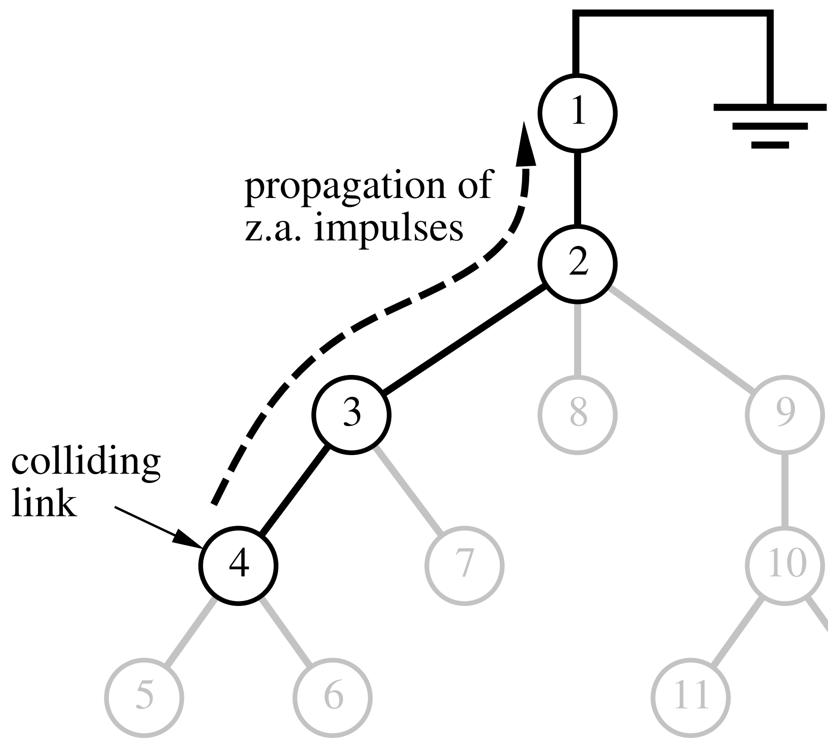

Figure 5.6: *The z.a. impulses need only be computed along the path from the colliding link to the base of the linkage. The gray links are not the colliding link or ancestors of it; for these links, the articulated z.a. impulse is* $\hat{\mathbf{0}}$.

term of the summation in (5.6) is non-vanishing. Since $h \neq k$, $\hat{\mathbf{Y}}_h = \hat{\mathbf{0}}$ by Lemma 8. The simplified expression for the articulated z.a. impulse of link h, a proper ancestor of k, is:

$$\hat{\mathbf{Y}}_h^A = {}_h\hat{\mathbf{X}}_{i'} \left[\mathbf{1} - \frac{\hat{\mathbf{I}}_{i'}^A \hat{\mathbf{s}}_{i'} \hat{\mathbf{s}}_{i'}'}{\hat{\mathbf{s}}_{i'}' \hat{\mathbf{I}}_{i'}^A \hat{\mathbf{s}}_{i'}} \right] \hat{\mathbf{Y}}_{i'}^A, \tag{5.8}$$

where i' is the unique child of link h which is on the path from h to k.

Computing the articulated z.a. impulses is half of the work; it is also necessary to compute the instantaneous changes of velocity due to a collision impulse. This is also done by taking the limit of dynamic equations for articulated bodies. From (4.32),

$$\lim_{t_c \to 0} \int_0^{t_c} \ddot{q}_i \, dt = \lim_{t_c \to 0} \int_0^{t_c} \frac{Q_i - \hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A \, {}_i\hat{\mathbf{X}}_h \hat{\mathbf{a}}_h - \hat{\mathbf{s}}_i' \left(\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A \hat{\mathbf{c}}_i \right)}{\hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i} \, dt.$$

The finite magnitude Q_i and $\hat{\mathbf{c}}_i$ vanish in the limit; $\hat{\mathbf{I}}_i^A$, ${}_i\hat{\mathbf{X}}_h$, and $\hat{\mathbf{s}}_i$ are constants that can be pulled outside of the integral. The resulting change in link velocity is

$$\Delta\dot{q}_i = -\frac{\hat{\mathbf{s}}_i'}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}\left[\hat{\mathbf{I}}_i^A\, {}_i\hat{\mathbf{X}}_h\Delta\hat{\mathbf{v}}_h + \hat{\mathbf{Y}}_i^A\right], \tag{5.9}$$

where $\Delta\hat{\mathbf{v}}_h$ is the limit of the infinite acceleration $\hat{\mathbf{a}}_h$ integrated over an infinitesimal interval. The change in link i's spatial velocity is obtained by applying the same technique to (4.30):

$$\lim_{t_c\to 0}\int_0^{t_c}\hat{\mathbf{a}}_i\ dt = \lim_{t_c\to 0}\int_0^{t_c}\left({}_i\hat{\mathbf{X}}_h\hat{\mathbf{a}}_h + \ddot{q}_i\hat{\mathbf{s}}_i + \hat{\mathbf{c}}_i\right)dt.$$

Again dropping terms that vanish as $t_c \to 0$, and simplifying,

$$\Delta\hat{\mathbf{v}}_i = {}_i\hat{\mathbf{X}}_h\Delta\hat{\mathbf{v}}_h + \Delta\dot{q}_i\hat{\mathbf{s}}_i. \tag{5.10}$$

At this point, a procedure can be described for determining the instantaneous change in a link's velocity, when a spatial collision impulse $\hat{\mathbf{P}}_{\text{coll}}$ is applied to it:

1. Compute the articulated inertias of all links of the multibody.

2. Compute the articulated z.a. impulses of the chain beginning at the colliding link, and ending at the base link [Equations (5.7), (5.8)].

3. Propagate instantaneous changes in joint and link velocities, along the path beginning at the root link and ending at the colliding link [Equations (5.9), (5.10)].

Although most processing occurs along the path from the base link to the colliding link, the effects of the other links enter through the articulated inertias of the path links. An algorithm for implementing the strategy is shown in Figure 5.7. Step 1 is not included because articulated inertias of all links are assumed available from the last dynamic integration.

5.3.3 Computing $\mathbf{K}_i$

The `impulseResponse` algorithm of Figure 5.7 relates a spatial collision impulse to a spatial change in velocity. But the collision matrix $\mathbf{K}_i$ relates non-spatial quantities. Figure 5.8 depicts the body frame $\mathcal{F}_k$ of the colliding link, as well as the collision frame $\mathcal{F}_{\text{coll}}$. Let $\mathbf{R}$ be the 3×3 rotation matrix taking vectors in $\mathcal{F}_{\text{coll}}$ to vectors in $\mathcal{F}_k$, and let $\mathbf{r}$ be the vector from the origin of $\mathcal{F}_{\text{coll}}$ to the origin of $\mathcal{F}_k$, in $\mathcal{F}_k$'s coordinates. In frame $\mathcal{F}_{\text{coll}}$, there is an impulsive force $\mathbf{p}\in\mathbb{R}^3$ but no impulsive torque. In frame $\mathcal{F}_k$, there is an impulsive force $\mathbf{Rp}$ and an impulsive torque $-\mathbf{r}\times\mathbf{Rp}$, or in spatial notation,

$$\hat{\mathbf{p}}_{\text{coll}} = \begin{bmatrix} \mathbf{R}^T & \mathbf{0} \\ \tilde{\mathbf{r}}\mathbf{R}^T & \mathbf{R}^T \end{bmatrix}\begin{bmatrix} \mathbf{p} \\ \mathbf{0} \end{bmatrix}. \tag{5.11}$$

```
impulseResponse
```

$\hat{\mathbf{Y}}_k^A \leftarrow -\hat{\mathbf{p}}_{\texttt{coll}}$

$i \leftarrow k$

`while link` i `has a parent`

  $h \leftarrow$ `index of parent of link` i

  $\hat{\mathbf{Y}}_h^A = {}_h\hat{\mathbf{X}}_i \left[\mathbf{1} - \dfrac{\hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i \hat{\mathbf{s}}_i'}{\hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i} \right] \hat{\mathbf{Y}}_i^A$

  $i \leftarrow h$

$\hat{\mathbf{a}}_0 \leftarrow \hat{\mathbf{0}}$

$h \leftarrow 0$

`repeat`

  $i \leftarrow$ `index of child of link` h `on path to link` k

  $\Delta \dot{q}_i = -\dfrac{\hat{\mathbf{s}}_i'}{\hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i} \left[\hat{\mathbf{I}}_i^A \, {}_i\hat{\mathbf{X}}_h \Delta \hat{\mathbf{v}}_h + \hat{\mathbf{Y}}_i^A \right]$

  $\Delta \hat{\mathbf{v}}_i = {}_i\hat{\mathbf{X}}_h \Delta \hat{\mathbf{v}}_h + \Delta \dot{q}_i \hat{\mathbf{s}}_i$

  $h \leftarrow i$

`until` $i = k$

Figure 5.7: `impulseResponse`. Compute $\Delta \hat{\mathbf{v}}_k$, the instantaneous change in spatial velocity of link k, when it experiences a collision impulse $\hat{\mathbf{p}}_{\text{coll}}$. This routine assumes that the articulated inertias of all links are already computed.

The 6×6 matrix is just the spatial transformation ${}_k\hat{\mathbf{X}}_{Coll}$. This equation shows how the non-spatial impulse $\mathbf{p}$ is mapped into a spatial impulse $\hat{\mathbf{p}}_{\text{coll}}$. The spatial change in velocity $\Delta \hat{\mathbf{v}}_k = (\Delta \boldsymbol{\omega}_k, \Delta \mathbf{v}_k)^T$ computed by `impulseResponse` must then be converted into a non-spatial change in velocity at the contact point. Since $\boldsymbol{\omega}_k$ and $\mathbf{v}_k$ are expressed in frame $\mathcal{F}_k$, the change in velocity at the contact point is

$$\begin{aligned} \Delta \mathbf{u} &= \mathbf{R}^T \left(\Delta \mathbf{v}_k + \Delta \boldsymbol{\omega}_k \times (-\mathbf{r}) \right) \\ &= \mathbf{R}^T \Delta \mathbf{v}_k + \mathbf{R}^T \tilde{\mathbf{r}} \Delta \boldsymbol{\omega}_k . \end{aligned}$$

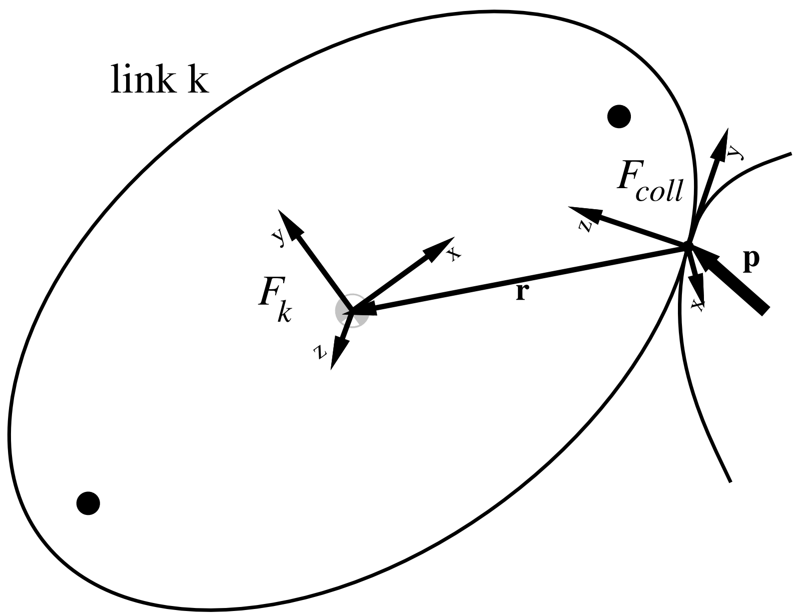

Figure 5.8: *The body frame of a colliding link and the collision frame.*

This is just the lower 3×1 component of the spatial product

$$\begin{bmatrix} \mathbf{R}^T & \mathbf{0} \\ \mathbf{R}^T \tilde{\mathbf{r}} & \mathbf{R}^T \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{\omega}_k \\ \Delta \mathbf{v}_k \end{bmatrix}. \tag{5.12}$$

(The upper 3×1 component of the product is the change in *angular* velocity at the contact point, which is not used in the collision model.) It is easily verified that the spatial transformation matrix in the above equation is ${}_{coll}\hat{\mathbf{X}}_k$, the spatial transformation from $\mathcal{F}_k$ to $\mathcal{F}_{\text{coll}}$.

The above observations, along with the algorithm `impulseResponse` provide a way to compute $\Delta \mathbf{u} \in \mathbb{R}^3$, the change in contact point velocity, given $\Delta \mathbf{p} \in \mathbb{R}^3$, an impulse applied at the contact point. But the matrix $\mathbf{K}_i$ relates changes in velocity to an arbitrary collision impulse. Since velocity changes at the contact point are linear in the applied collision impulse it is only necessary to compute the response to a set of basis impulses at the collision point. Let $\phi : \mathbb{R}^3 \to \mathbb{R}^3$ be the mapping from applied impulse to change in contact point velocity. An arbitrary impulse $\mathbf{p}$ may be decomposed as

$$\mathbf{p} = p_x \mathbf{e}_x + p_y \mathbf{e}_y + p_x \mathbf{e}_z,$$

where $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_z$ are the canonical basis vectors of the collision frame. Then

$$\phi(\mathbf{p}) = p_x \phi(\mathbf{e}_x) + p_y \phi(\mathbf{e}_y) + p_z \phi(\mathbf{e}_z).$$

By building the matrix $\mathbf{K}_i$ as

$$\mathbf{K}_i = \left[\begin{array}{c|c|c} \\ \phi(\mathbf{e}_x) & \phi(\mathbf{e}_y) & \phi(\mathbf{e}_z) \\ \\ \end{array} \right], \tag{5.13}$$

it follows that

$$\phi(\mathbf{p}) = \mathbf{K}_i \ \mathbf{p} \tag{5.14}$$

for any $\mathbf{p} \in \mathbb{R}^3$. Thus, the impulse response of the colliding body may be completely characterized by the impulse response to three test impulses, each a unit vector in one of the canonical basis directions.

Finally, the algorithm for computing $\mathbf{K}_i$ may be completely written. It is shown in Figure 5.9. After $\mathbf{K}_1$ and $\mathbf{K}_2$ are computed, the complete multibody collision matrix is given by their sum. Once $\mathbf{K}$ is known, a collision integration can be performed to determine the collision impulse; whether either of the colliding bodies are multibodies or simple rigid bodies, all of the necessary dynamic information is encoded in $\mathbf{K}$.

5.3.4 Propagating impulses through multibodies

After the collision impulse $\mathbf{p}$ is determined, it must be propagated through the multibody to determine the instantaneous change in joint velocities. The machinery to do this has already been developed. The algorithm `propagateImpulse` (Figure 5.10) is similar to the `impulseResponse` algorithm, except that the effects are propagated throughout the entire tree linkage rather than just along the path to the colliding link. The iteration is performed by enumerating over the joints in increasing order, as in the forward dynamics algorithms of Chapter 4.

Figure 5.11 summarizes the entire process of resolving a collision that involves a link of a multibody. When and n_1-link and an n_2-link two multibody collide, the

```
compMultibodyKi

    compute the articulated inertias for all links of the multibody
    compute _k X̂_coll and _coll X̂_k from R and r
    for i = 1 to 3
        p̂_coll ← _k X̂_coll [e_i ; 0]
        call impulseResponse to compute Δv_k
        Δφ_i ← lower 3×1 component of _coll X̂_k Δv̂_k
    K_i ← [ Δφ_1 | Δφ_2 | Δφ_3 ]
```

compute the articulated inertias for all links of the multibody

compute ${}_k\hat{\mathbf{X}}_{coll}$ and ${}_{coll}\hat{\mathbf{X}}_k$ from $\mathbf{R}$ and $\mathbf{r}$

for $i = 1$ to 3

$\hat{\mathbf{p}}_{\texttt{coll}} \leftarrow {}_k\hat{\mathbf{X}}_{coll} \begin{bmatrix} \mathbf{e}_i \\ \mathbf{0} \end{bmatrix}$

call impulseResponse to compute $\Delta \mathbf{v}_k$

$\Delta \boldsymbol{\phi}_i \leftarrow$ lower 3×1 component of ${}_{coll}\hat{\mathbf{X}}_k \Delta \hat{\mathbf{v}}_k$

$$\mathbf{K}_i \leftarrow \left[\begin{array}{c|c|c} \Delta \boldsymbol{\phi}_1 & \Delta \boldsymbol{\phi}_2 & \Delta \boldsymbol{\phi}_3 \end{array} \right]$$

Figure 5.9: `compMultibodyKi`. Compute $\mathbf{K}_i$, half of the collision matrix corresponding to one of the bodies in the collision. Inputs are k, the index of the colliding link, and $\mathbf{R}$ and $\mathbf{r}$, which specify the relative positions of the collision and body frame.

total cost of processing the collision is $O(n_1 + n_2 + C)$, where C is the cost of performing a collision integration. A linear time multibody collision response algorithm allows the efficient simulation of large articulated structures in an impulse-based setting.

For floating linkages, the same adjustment that was made for the forward dynamics algorithm must be made in the collision response algorithms. No change is required for articulated inertia and z.a. impulse propagation. When computing velocity changes, however, the base link must be handled as a special case. Since there is no inboard impulse applied to the base link,

$$\Delta \hat{\mathbf{v}}_1 = -\left(\hat{\mathbf{I}}_1^A\right)^{-1} \hat{\mathbf{Y}}_1^A.$$

For all joints and other links, velocity change propagation needs no adjustment.

With simple rigid bodies, the collision matrix is always positive definite, however, the multibody collision matrix is only guaranteed to be positive semidefinite. Thus, the invertibility of $\mathbf{K}$ becomes an issue. Figure 5.12 shows an example where $\mathbf{K}$ can be singular.

```
propagateImpulse
```

$\hat{\mathbf{Y}}_k^A \leftarrow -\hat{\mathbf{p}}_{\texttt{coll}}$

$i \leftarrow k$

`while link` i `has a parent`

$\quad h \leftarrow$ `index of parent of link` i

$$\hat{\mathbf{Y}}_h^A = {}_h\hat{\mathbf{X}}_i \left[\mathbf{1} - \frac{\hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i \hat{\mathbf{s}}_i'}{\hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i} \right] \hat{\mathbf{Y}}_i^A$$

$\quad i \leftarrow h$

`for all links` i `not on path from base link to link` k`,` $\hat{\mathbf{Y}}_i^A \leftarrow \hat{\mathbf{0}}$

$\hat{\mathbf{a}}_0 \leftarrow \hat{\mathbf{0}}$

`for` $i = 1$ `to` n

$\quad h \leftarrow$ `index of link inboard to joint` i `/* & link` i `is outboard */`

$$\Delta \dot{q}_i = -\frac{\hat{\mathbf{s}}_i'}{\hat{\mathbf{s}}_i' \hat{\mathbf{I}}_i^A \hat{\mathbf{s}}_i} \left[\hat{\mathbf{I}}_i^A \, {}_i\hat{\mathbf{X}}_h \Delta \hat{\mathbf{v}}_h + \hat{\mathbf{Y}}_i^A \right]$$

$$\Delta \hat{\mathbf{v}}_i = {}_i\hat{\mathbf{X}}_h \Delta \hat{\mathbf{v}}_h + \Delta \dot{q}_i \hat{\mathbf{s}}_i$$

Figure 5.10: `propagateImpulse`. Compute the instantaneous change in spatial velocity of joints of a tree linkage, when link k experiences a spatial collision impulse $\hat{\mathbf{p}}_{\text{coll}}$. $\hat{\mathbf{p}}_{\text{coll}}$ is computed from $\mathbf{p}$ via Equation (5.11). This algorithm assumes that the articulated inertias of all links are already computed.

The two-link multibody falls and collides with the fixed horizontal plate. Because there are only two degrees of freedom in the multibody motion, no matter what impulse $\mathbf{p}$ is applied at the contact point, the total change in relative contact point velocity $\Delta\mathbf{u}$ can have no component in the y direction. Since $\Delta\mathbf{u} = \mathbf{K}\mathbf{p}$, the 3×3 matrix $\mathbf{K}$ only has rank two. But the collision integration algorithms require $\mathbf{K}^{-1}$. This is not a problem in practice. Let $\tilde{\mathbf{K}}^{-1}$ be the canonical "inverse" of $\mathbf{K}$ obtained by singular value decomposition (SVD) [PTVF92]. Then $\tilde{\mathbf{K}}^{-1}\Delta\mathbf{u}$ gives the minimum norm $\mathbf{p}$ that causes the given $\Delta\mathbf{u}$; the $\mathbf{p}$ computed in this manner has no component in the null space of $\mathbf{K}$. This gives the expected results: in the

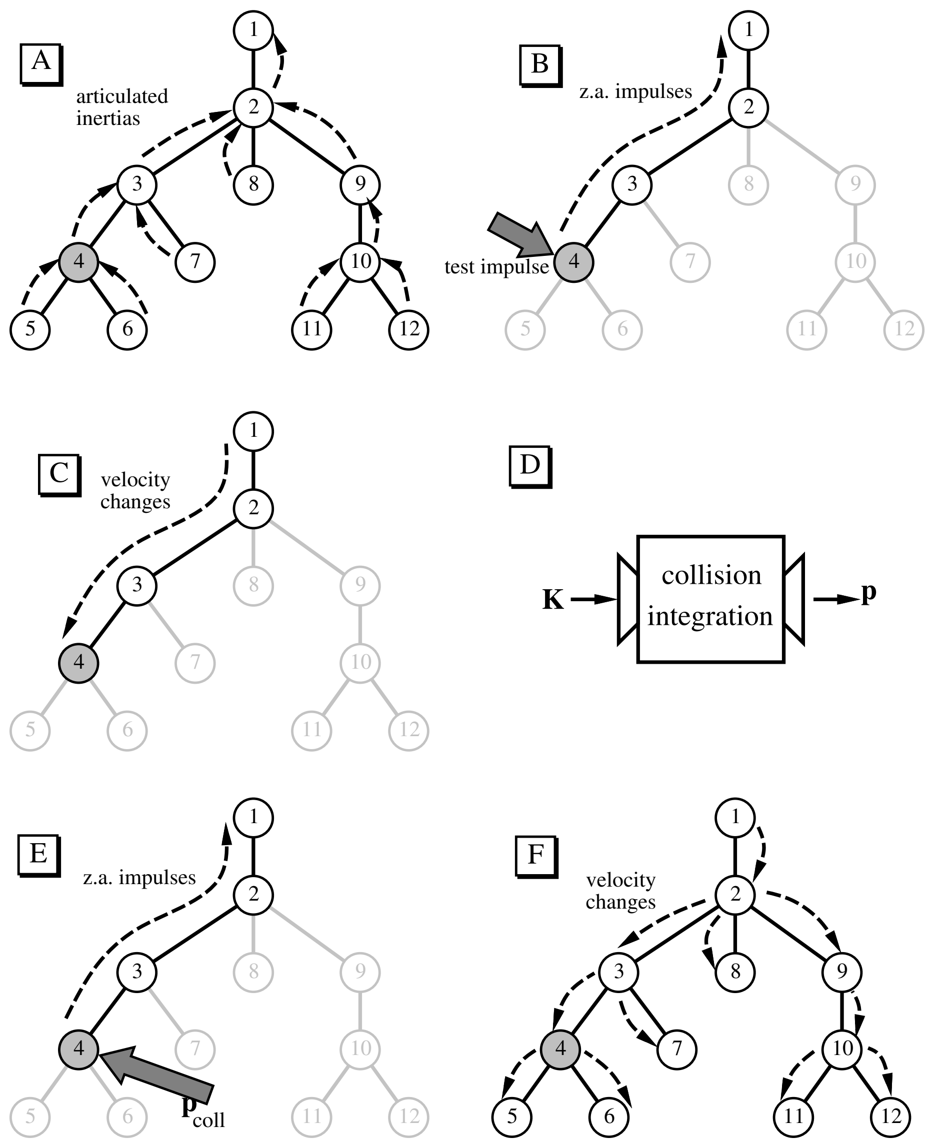

Figure 5.11: *The entire procedure for resolving collisions involving a multibody; here, link* 4 *is the colliding link. (A) Propagate articulated inertias. This step can be eliminated by saving inertias from the forward dynamics algorithm. (B) Apply test impulse to the colliding link, and propagate z.a. impulses up to the base link. (C) Propagate joint and link velocity changes back to the colliding link. Steps B and C are performed three times, for test impulses in the x, y, and z directions; afterwards,* $\mathbf{K}_1$ *is known.* $\mathbf{K}_2$ *is found similarly, and these are combined to give* $\mathbf{K}$*. (D) Collision integration determines* $\mathbf{p}$*. (E) Propagate z.a. impulses from* $\hat{\mathbf{p}}_{coll}$ *up to base link. (F) Propagate velocity changes throughout the tree.*

collision of Figure 5.12, the collision impulse will lie in the x-z plane. The same approach can be used when the collision matrix has rank one.

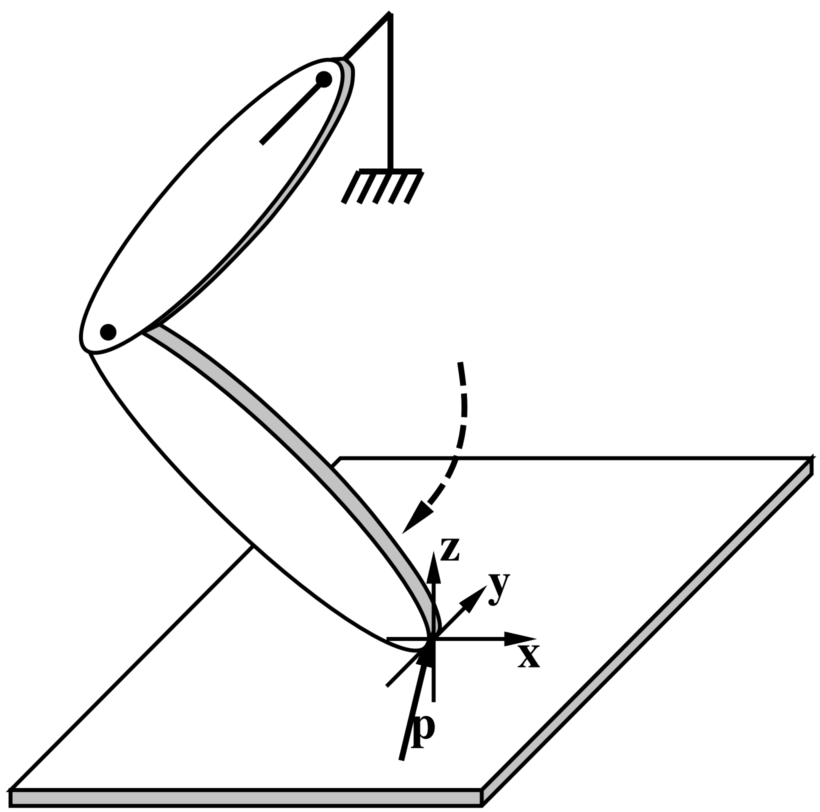

Figure 5.12: *A situation in which the multibody collision matrix is rank deficient.*

5.4 Supporting control systems

To animate multibodies, it is usually necessary to apply control forces or torques at the joints in order to achieve the desired physical motion. (For the rest of this section, *force* is used in a generalized sense to mean force or torque, depending on whether a joint is prismatic or revolute.) Using the collision check scheduler that is already present in an impulse-based simulator, a flexible control architecture can be built on top of the dynamic simulator.

5.4.1 Types of controllers

There are several types of controllers that can be used to control multibodies. More information than appears here can be found in [Cra89]. Many systems contain passive elements such as springs and dampers attached at the joints of a multibody. These are not controllers in a strict sense, however they influence the dynamics of the system in a similar way. They may be undesirable but unavoidable characteristics of the system, such as friction in joint bearings, or they may be elements intentionally added to the system by a designer, such as the shock-absorbers on a car. Passive controllers are characterized by the force f applied, given the position q and velocity $\dot{q}$ of a joint, such as:

$$
\begin{aligned}
f &= -\mu \ \mathrm{sgn}(\dot{q}) \\
f &= -\rho \ \dot{q} \\
f &= -k \ (q - q_n) \\
f &= -\frac{c}{q - q_n}
\end{aligned}
$$

In order, these equations correspond to: constant friction, viscous damping, a linear spring, and a hard spring which arises, for example, in a sealed, prismatic cylinder (this latter force has been exploited for hopping robot control [Rai86]).

Passive controllers are most accurately modeled by placing the force computations within the inner loop of the dynamic integration. For example, every time the integrator calls `treeFwdDyn` to compute $\ddot{\mathbf{q}}$, passive controllers forces are recomputed, based on $\mathbf{q}$ and $\dot{\mathbf{q}}$. These control forces may be lumped in with actuator forces, $\mathbf{Q}$, since they modify the dynamics in the same way. The "control laws" for these passive elements are simple, so even in the inner integration loop, the cost of recomputing the control forces is a negligible part of computing the forward dynamics.

At the lowest level of a control system are elements which control external forces to be applied at joints in order to maintain desired joint positions or velocities. Proportional-derivative (PD) or proportional-integral-derivative (PID) controllers are two common types. PD controllers are used, for example, in the human athlete models of Hodgins, *et. al.* [HWBO95]. The basic operation of a low level controller is depicted in Figure 5.13.

Suppose a particular joint is to be maintained at some desired position, q_{des}. One approach would be to attach a spring with natural length q_{des} to the joint. The stiffer the spring, the harder the joint would be forced to the desired value. To prevent oscillations, a damper could also be added to the system. A positional PD controller mimics the behavior of this spring-damper system. In control parlance, the spring constant and damping constant are called k_p and k_v, respectively (p for position and v for velocity). The actuator force calculation for a positional PD controller is given by

$$Q = -k_p(q - q_{\mathrm{des}}) - k_v\dot{q}.$$

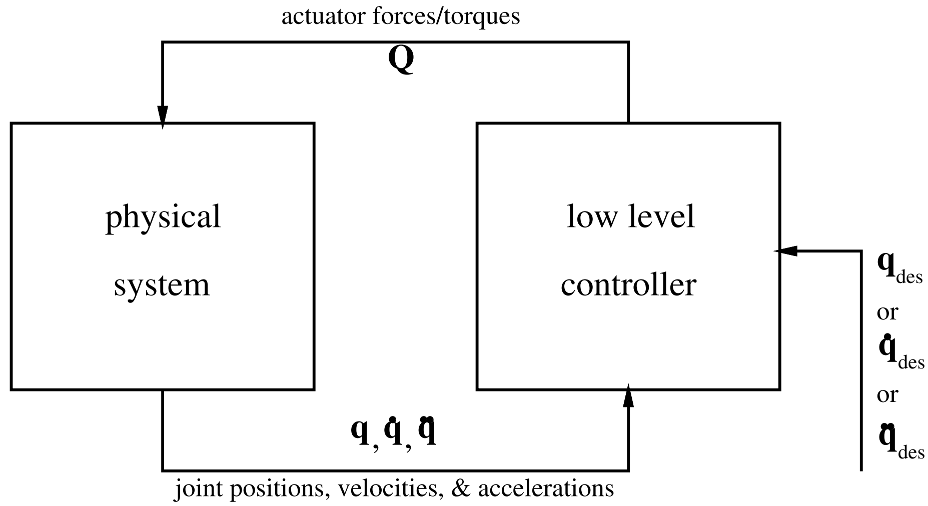

Figure 5.13: *The coupling of a low level controller to a multibody system. The controller takes as inputs the position, velocity, and possibly acceleration of a particular joint, and computes a torque to be applied at that joint, based on the control law and desired motion.*

In some cases, a PD controller is insufficient for maintaining q_{des}. Consider the prismatic joint in Figure 5.14. It is easily verified that the external force results in an equilibrium position of

$$q_{\text{des}} - \frac{mg}{k_p}.$$

The static error is made arbitrarily small by choosing a very large k_p, however this can lead to instability in the system, and numerical stiffness in the equations. Instead, a proportional-derivative-integral (PID) controller can be used. This type of controller augments the spring and damper components with an integral term, that is based on the static error accumulated over time:

$$Q(t) = -k_p(q(t) - q_{\text{des}}) - k_v\dot{q}(t) - k_i \int_0^t \left(q(\tau) - q_{\text{des}}\right)\, d\tau. \tag{5.15}$$

A PID controller provides eventual convergence to the q_{des} in cases such as the one in Figure 5.14, although they are more difficult to analyze and introduce new stability concerns.

Another type of low level controller is a computed torque controller, which shields a higher level controller from the internal dynamics of a system. The dynamics of a point mass are trivial. To make the mass's acceleration equal to $\mathbf{a}$, a force $\mathbf{f} = m\mathbf{a}$ is applied. For a multibody, the dynamics are more complicated, and joint accelerations are not proportional to joint forces. A computed torque controller hides this complexity, so that a higher level controller can directly specify the desired accelerations of the system. If the desired joint

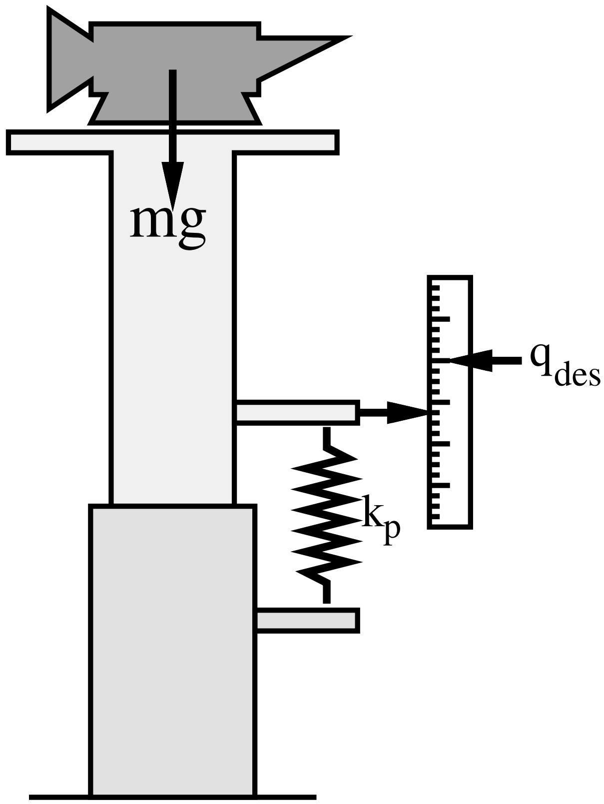

Figure 5.14: *A PD controller with position constant k_p causes the unweighted joint to attain a value of q_{des}, but it can not do so when the joint is loaded with a static weight.*

accelerations of a multibody system are $\ddot{\mathbf{q}}_{\text{des}}$, the computed torque law is

$$\mathbf{Q} = \mathbf{H}(\mathbf{q})\ddot{q}_{\text{des}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \; + \mathbf{G}(\mathbf{q}) - \mathbf{J}^T(\mathbf{q})\ \hat{\mathbf{f}}. \tag{5.16}$$

Substituting into Equation (5.1) shows that with these joint forces, $\ddot{q} = \ddot{q}_{\text{des}}$. A computed torque controller requires much more computation than PD and PID controllers, since a dynamic model of the multibody must be analyzed to determine the joint forces.

Unlike passive elements like springs, controllers run at finite rates. A controller is updated with new values of $\mathbf{q}$ and $\dot{\mathbf{q}}$ at regular intervals, and it commands actuator forces which will persist until the next update, even though $\mathbf{q}$ and $\dot{\mathbf{q}}$ change over the interval. The update rate must be fast enough so that the controller has reasonably current information, otherwise instability results. The required rate depends roughly on the mass of the system and on the controller gains. Less massive systems and higher gains require higher update rates.

High level controllers are needed to give multibody systems intelligent behavior. While a low level controllers might allow a human figure to stand, a high level controller is necessary to make it walk toward a goal. High level controllers are less concerned with the dynamic stability of the system, and more concerned with the overall behavior. They run at rates slower than the rates of the low level controllers. For example, a robotic controller that examines machine vision input would have a fairly slow update rate. High level controllers are more varied than low level controllers, and there are few canonical types. Finite state machine controllers have enjoyed recent success as high level controllers for hopping robots [Rai86] and models of human athletes [HWBO95]. These track the phase of the overall system (such as flight or loading), and send commands to low level PD controllers based on the phase. Use of state machines for higher level behavior control is described by Brooks in the context of robot planning [Bro86], and by Ahmad *et. al.* for scenario control [ACH$^+$94].

5.4.2 Controller scheduling

There may be many different controllers for a system, running in parallel at different rates. In simulation, controllers are software routines that compute applied forces from the current state. These routines must be called to recompute the control forces at a rate matching that of the real controller they model. Scheduling control update events is simple in an impulse-based simulator. As discussed in Chapter 2, the integration intervals are determined by the collision heap. The system is integrated to the point in time indicated by the TOI field of the top heap element, at which point a collision check is performed between two bodies. The simulator guarantees that no heap events are missed.

Control update events can be added to the collision check events in the heap. Suppose a PD controller is to update joint torques at a 1 kHz rate. A structure for this control event is placed into the heap, with a time field set to the current time plus 0.001 seconds. Using the same scheduling algorithm, the simulation will be stopped at the appropriate time, and the PD update procedure can be called. The control event is placed back into the heap, with the time field incremented by 0.001 s. In this way, the simulator can manage a variety controllers running independently and at different rates. Adding control update events to the heap does not significantly affect the size of the integration steps; collision check events still dominate the top of the heap. For simulations with a only few bodies, collision checks can be made at rates on the order of 10 kHz. This is far higher than typical controller frequencies. The passive controller forces are computed within the inner integration loop, so they are not scheduled in the heap.

5.4.3 *Impulse's* control support architecture

Impulse shields the designer of a control system from the inner workings of the control law scheduler and the dynamic integration loop through a layer of interface routines. Standard types of low level controllers are provided. The designer writes high level control routines as C functions that are compiled and linked with the rest of the simulator code. When a high level controller is instantiated, it is registered with the scheduler so that it may be called back at the desired rate. The high level control routine can instantiate low level controllers and interact with them. *Impulse's* control library contains:

1. Routines to create passive elements, and attach them to specified joints of a multibody. Parameters like spring constants and damping coefficients are user specified.

2. Routines to create low level controllers. PD and PID controllers are available, in both position- and velocity-control varieties. Computed torque controllers are also provided. The rates and gains of these controllers are specified when they are created. Communication with these controllers is handled through routines that command desired positions, velocities, or accelerations to an existing controller, and disable or enable the controller. A high level controller typically calls these interface routines.

3. Routines to execute general high level controllers at specified rates through a call back mechanism. The high level controller routines (currently C functions) are written by the designer. When called, these high level controller routines are passed a pointer to the object being controlled, so that the same control function can be used for many clones of a particular object.

With these library routines, one can build up interesting control systems with a few lines of C code. Figure 5.15 is a system level view of the control support architecture. The lowest level is the simulation engine, which contains modules to compute the dynamics of rigid bodies and multibodies, and to perform collision detection and collision response between bodies. On top of this layer is the control library described above. The designer specifies high level controllers on top of the library. A high level controller may do nothing more than instantiate some low level controllers, and send them initial commanded values.

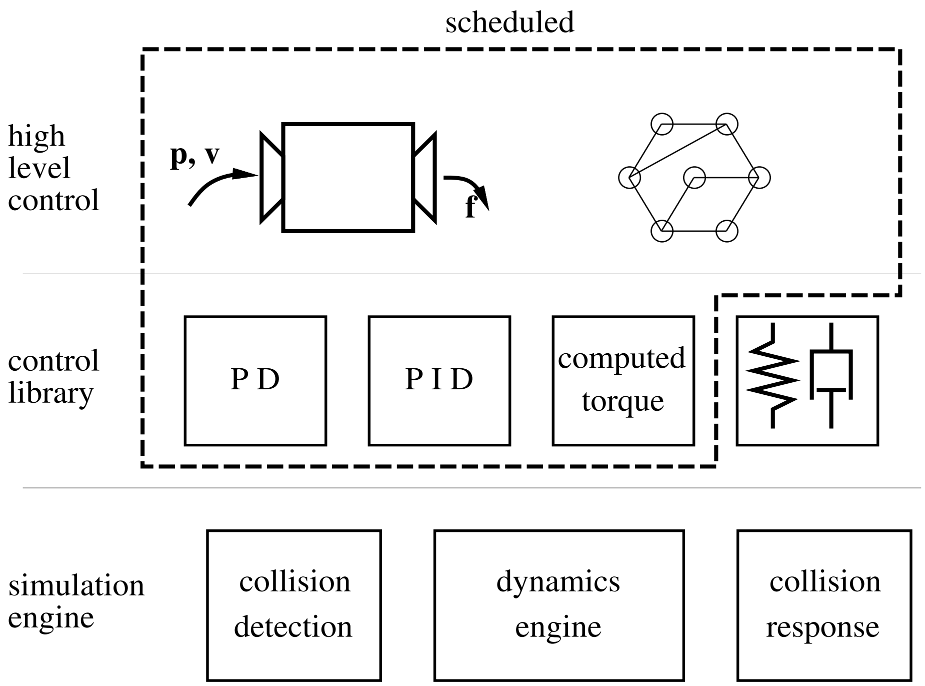

Figure 5.15: *The control support architecture of* Impulse. *Lower level controllers are provided on top of the simulation engine, and the designer writes the high level controllers. The dotted line indicates controllers that are scheduled in the heap.*

Or it can perform complex computations involving the state of the system and environment to determine behavior. The designer is free to create new types of low level controllers, or a control layer between the levels shown. Examples that use this control support architecture are described in Chapter 7.

Chapter 6

Computation of Mass Properties for Polyhedral Bodies

For dynamic simulation involving rigid bodies, it is necessary to know several parameters describing the mass distribution of these bodies. A common formulation comprises ten parameters: the total mass (a scalar), the location of the center of mass (three parameters), and the moments and products of inertia about the center of mass (six parameters). One can always find a body frame, with origin at the body's center of mass and axes aligned with its principle axes of inertia. In this way, the entire mass distribution can be described with a reduced set of four parameters, however the larger parameterization is still needed as a starting point.

The problem of computing mass properties of solid objects has been studied extensively. Lee and Requicha note that the algorithms are closely tied to the underlying representation of solid objects, and give an excellent survey of the various families of algorithms in existence [LR82a]. Representations based upon primitive instancing and simple sweeping have small domain of application; only simple "canned" objects may be described. Algorithms based on decomposition fall into two categories, according to whether the decomposition is exact or approximate. Exact decompositions are difficult to obtain for general objects and usually require considerable human labor [LK84]. Approximate decompositions, such as spatial enumerations or octree methods may be easier to generate but yield only estimates of the desired integral properties [LR82b]. The resolution of these representations determine the accuracy of the estimate, and errors in the representation tend to dominate numerical errors [LR82a]. Algorithms that operate directly on CSG representations are computationally expensive; indeed, the best algorithms for CSG operate by first converting the CSG representations into approximate cell decompositions. Integrals over solid regions can also be evaluated using Monte Carlo techniques, based on sampling points inside a volume containing the solid. Accuracy increases with the number of sample points, but convergence is slow, and long execution times are required for accurate results [LR82a].

Algorithms based on boundary representations (Breps) remain the method of choice for exact, efficient computation of integrals over solids. This is fortunate since most graphical rendering algorithms require Breps, as do many efficient collision detection algorithms [Lin93, Van91]. Thus, for applications such as dynamic simulation, Breps are likely to be available. Lien and Kajiya give an algorithm for computing integrals over arbitrary nonconvex polyhedra, based on a Brep [LK84]. It is $O(n)$ in the polyhedron complexity, and works by projecting triangular faces to the origin, forming tetrahedra, and summing the contributions of these tetrahedra to the composite integrals. Numerical errors increase in near degenerate situations where the tetrahedra are long and thin. These errors are reduced by shifting the projection origin to the barycenter of the object, however tetrahedra of high aspect ratio can still occur, especially for highly tessellated objects. Also, the algorithm essentially triangulates any non-triangular faces of the original polyhedron, and the manner in which this is done may also generate long, thin tetrahedra. Nevertheless, the algorithm proposed by Lien and Kajiya is fairly easy to code and generalizes to higher dimensions. It may be easily adapted to compute any integral of the form $x^i y^j z^k$ over the solid region, in addition to the integrals arising when computing mass properties.

This chapter presents an exact algorithm for computing mass properties of uniform density polyhedra, based on a boundary representation of the solid. Like Lien and Kajiya's algorithm, it is exact, and linear in the number of vertices, edges, or faces of the polyhedron. The polyhedron may be nonconvex, and the faces may have any number of sides. The only assumptions made on the representation are that one can enumerate over all faces of the boundary, and that given a face, one can enumerate over all edges in counterclockwise order. In contrast to Lien and Kajiya's algorithm, the algorithm is optimized for computing the mass properties required for dynamic simulation, namely mass, center of mass, and moments and products of inertia. These quantities are all computed in parallel during a single traversal of the polyhedron, so that common subexpressions are exploited; it is very fast. In addition, the algorithm presented here adaptively changes the projection direction, minimizing degeneracy and reducing numerical errors over those in Lien and Kajiya's and other algorithms.

6.1 Rigid-body mass parameters

See Appendix A.3 for an overview of rigid body dynamics. Key quantities in rigid-body dynamics are a body's mass m, center of mass $\mathbf{r}$ and inertia tensor $\mathbf{I}_o$, also called the mass matrix. In order to formulate the equations of motion for the body, these quantities must be determined. The initial problem may be expressed as follows:

Problem 9 Given: *A rigid body comprising N parts, $B_1, \ldots, B_N$, each a uniform density polyhedron. There are no restrictions on the convexity or genus of the polyhedra, nor on the shape of the bounding faces. For each polyhedron B_i, either its density ρ_i or mass m_i is specified, and the geometries of all of the polyhedra are specified relative to a single reference frame.* Compute: *The mass m, and the reference frame coordinates of the center of mass $\mathbf{r}$ and inertia tensor $\mathbf{I}_o$ for the entire rigid body.*

The mass m_i and density ρ_i of polyhedral part B_i are related by $m_i = \rho_i V_i$, where V_i is the volume of the polyhedron. Assuming one can compute

$$V_i = \int_{B_i} dV,$$

the masses and densities of each polyhedron can be found. The total mass is $m = \sum_{i=1}^{N} m_i$. The coordinates of the center of mass $\mathbf{r}$ for the entire body are

$$\mathbf{r} = \frac{1}{m} \sum_{i=1}^{N} \left(\int_{B_i} x \; dm, \; \int_{B_i} y \; dm, \; \int_{B_i} z \; dm \right)^T .$$

The integrals above are the sums over all differential mass elements of the body, but for uniform density polyhedra, these are easily converted to volume integrals. For body B_i, $dm = \rho_i \; dV$. Thus,

$$\mathbf{r} = \frac{1}{m} \sum_{i=1}^{N} \rho_i \left(\int_{B_i} x \; dV, \; \int_{B_i} y \; dV, \; \int_{B_i} z \; dV \right)^T .$$

The inertia tensor is a symmetric matrix, containing the moments and products of inertia:

$$\mathbf{I}_0 = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{zx} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{yz} & I_{zz} \end{bmatrix} .$$

The moments of the inertia are given by

$$\begin{aligned}
I'_{xx} = \sum_{i=1}^{N} \int_{B_i} (y^2 + z^2)\ dm &= \sum_{i=1}^{N} \rho_i \int_{B_i} (y^2 + z^2)\ dV \\
I'_{yy} = \sum_{i=1}^{N} \int_{B_i} (z^2 + x^2)\ dm &= \sum_{i=1}^{N} \rho_i \int_{B_i} (z^2 + x^2)\ dV \\
I'_{zz} = \sum_{i=1}^{N} \int_{B_i} (x^2 + y^2)\ dm &= \sum_{i=1}^{N} \rho_i \int_{B_i} (x^2 + y^2)\ dV.
\end{aligned}$$

Finally, the products of inertia are given by

$$\begin{aligned}
I'_{xy} = \sum_{i=1}^{N} \int_{B_i} xy\ dm &= \sum_{i=1}^{N} \rho_i \int_{B_i} xy\ dV \\
I'_{yz} = \sum_{i=1}^{N} \int_{B_i} yz\ dm &= \sum_{i=1}^{N} \rho_i \int_{B_i} yz\ dV \\
I'_{zx} = \sum_{i=1}^{N} \int_{B_i} zx\ dm &= \sum_{i=1}^{N} \rho_i \int_{B_i} zx\ dV.
\end{aligned}$$

The primed inertia quantities above are not exactly the values appearing in the inertia tensor. The primed values are computed relative to the given reference frame, but the values in the inertia tensor must be computed relative to a frame with origin at the center of mass. One way to accomplish this is to first compute the location of the center of mass in the given frame, and then to apply a translation to the body which brings the center of mass to the origin. After performing this transformation, the primed quantities can be directly inserted into the inertia tensor.

A better solution is to use the *transfer-of-axis* relations for transferring a moment or product of inertia about one axis to a corresponding one about a parallel axis. To transfer the primed values computed above to a frame at the center of mass, one uses [MK86]:

$$\begin{aligned}
I_{xx} &= I'_{xx} - m(r_y^2 + r_z^2) & \qquad I_{xy} &= I'_{xy} - m r_x r_y \\
I_{yy} &= I'_{yy} - m(r_z^2 + r_x^2) & \qquad I_{yz} &= I'_{yz} - m r_y r_z \\
I_{zz} &= I'_{zz} - m(r_x^2 + r_y^2) & \qquad I_{zx} &= I'_{zx} - m r_z r_x.
\end{aligned}$$

The unprimed quantities are inserted into the inertia tensor. If the transfer-of-axis relations are used, it is not necessary to explicitly transform the vertices of the polyhedron after computing the center of mass, hence all of the integrals can be computed simultaneously.

Although not explicitly allowed in the problem statement, point masses placed at various locations on the rigid body are easily handled. Suppose the component body B_i is a point mass rather than a polyhedron, located at position $\mathbf{r}_i$ in the reference frame. All of the required mass integrals can still be computed, since for a point mass

$$\int_{B_i} f \; dm = m_i f(\mathbf{r}_i)$$

where f is any scalar function of x, y, and z. This substitution may be used to compute all integrals over B_i.

Rigid body dynamics can be computed more efficiently with a reduced set of mass properties, based on a body frame. This is a coordinate system attached to the body, with origin at the body's center of mass, and axes aligned with the body's principle axes of inertia. In this case, all of the products of inertia vanish and the inertia tensor is diagonal. To compute the body frame, one must first compute $\mathbf{r}$ and $\mathbf{I}_o$ as above, and then find and a rotation matrix $\mathbf{R}$ such that

$$\mathbf{I} = \mathbf{R}^T \mathbf{I_o} \mathbf{R}$$

is diagonal. In this case, $\mathbf{I}$ is the diagonalized mass matrix, and $\mathbf{R}$ is the rotation matrix transforming vectors in the body frame to vectors in the original reference frame; the columns of $\mathbf{R}$ are the $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$ unit vectors of the body frame, expressed in the reference coordinates. The diagonal elements of $\mathbf{I}$ are the moments of inertia about the principle axes of the body. Since these are positive, $\mathbf{I}$ and $\mathbf{I}_o$ are positive definite. Diagonalizing a matrix, that is finding $\mathbf{R}$, is a classical problem of linear algebra. The Jacobi method works quite well for this application since $\mathbf{I}_o$ is real, symmetric, and of moderate size. See [PTVF92] for a complete discussion of this technique as well as an implementation.

6.2 Derivation of the algorithm

From the previous section, all required mass properties can be found from ten integrals over volume, for each of the individual polyhedral components. To simplify notation, the polyhedron index is henceforth dropped, and only a single polyhedral body is considered. The domain of integration is written $\mathcal{V}$ as a reminder that it is a volume. The remainder of this chapter describes an efficient, exact algorithm for calculating these ten integrals:

$$
\begin{aligned}
T_1 &= \int_{\mathcal{V}} 1 \; dV & T_{x^2} &= \int_{\mathcal{V}} x^2 \; dV & T_{xy} &= \int_{\mathcal{V}} xy \; dV \\
T_x &= \int_{\mathcal{V}} x \; dV & T_{y^2} &= \int_{\mathcal{V}} y^2 \; dV & T_{yz} &= \int_{\mathcal{V}} yz \; dV \\
T_y &= \int_{\mathcal{V}} y \; dV & T_{z^2} &= \int_{\mathcal{V}} z^2 \; dV & T_{zx} &= \int_{\mathcal{V}} zx \; dV. \\
T_z &= \int_{\mathcal{V}} z \; dV
\end{aligned}
\tag{6.1}
$$

Note that each is an integral of a monomial in x, y, and z. The basic idea is to use the divergence theorem to reduce each of the volume integrals to a sum of surface integrals over the individual faces of the polyhedron. Each of these surface integrals are evaluated in terms of integrals over a planar projection of the surface. For polygons in the plane, Green's theorem reduces the planar integration to a sum of line integrals around the edges of the polygon. Finally, these line integrals are evaluated directly from the coordinates of the projected vertices of the original polyhedron. Figure 6.1 illustrates the approach: the volume integral is ultimately reduced to a collection of line integrals in the plane, and the values from these integrations are propagated back into the value of the desired volume integration.

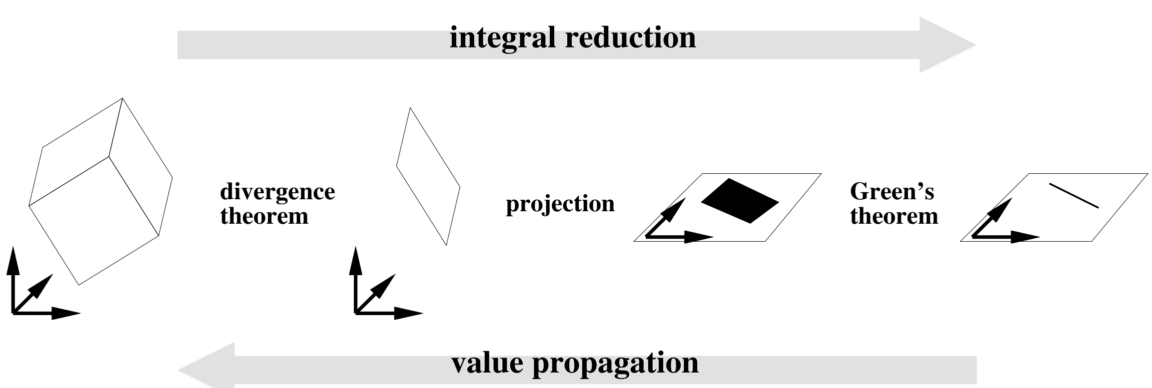

Figure 6.1: *The strategy for evaluating volume integrals. Complicated integrals are decomposed into successively simpler ones, and the values from evaluating the simplest integrals are combined and propagated back to evaluate the original ones.*

6.2.1 Reduction to surface integrals

The first reduction is from an integral over the three-dimensional polyhedral volume to a sum of integrals over its two-dimensional planar faces. This reduction is easily accomplished with the divergence theorem [Ste82]:

Theorem 20 (Divergence) *Let $\mathcal{V}$ be a region in space bounded by the surface $\partial\mathcal{V}$. Let $\hat{\mathbf{n}}$ denote the exterior normal of $\mathcal{V}$ along its boundary $\partial\mathcal{V}$. Then*

$$\int_{\mathcal{V}} \nabla \cdot \mathbf{F}\ dV = \int_{\partial\mathcal{V}} \mathbf{F} \cdot \hat{\mathbf{n}}\ dA \tag{6.2}$$

for any vector field $\mathbf{F}$ defined on $\mathcal{V}$.

For a polyhedral volume, the right-hand side of (6.2) can be broken up into a summation over faces of constant normal, so $\hat{\mathbf{n}}$ can be pulled outside the integral. The integrals to be computed, for example $\int_{\mathcal{V}} x\ dV$, do not immediately appear to be of the form in the theorem. But one can find many vector fields whose divergence is the function x; a particularly simple choice is $\mathbf{F}(x, y, z) = (\frac{1}{2}x^2, 0, 0)^T$. This choice has the added advantage that two of its components are identically zero, so that the dot product on the right-hand side of (6.2) becomes a scalar multiplication. By making similar choices for the other integrals and applying the divergence theorem, Equations (6.1) become:

$$\begin{aligned}
& & T_{x^2} &= \frac{1}{3}\sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_x \int_{\mathcal{F}} x^3\ dA \\
T_1 &= \sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_x \int_{\mathcal{F}} x\ dA & T_{y^2} &= \frac{1}{3}\sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_y \int_{\mathcal{F}} y^3\ dA \\
T_x &= \frac{1}{2}\sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_x \int_{\mathcal{F}} x^2\ dA & T_{z^2} &= \frac{1}{3}\sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_z \int_{\mathcal{F}} z^3\ dA \\
T_y &= \frac{1}{2}\sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_y \int_{\mathcal{F}} y^2\ dA & T_{xy} &= \frac{1}{2}\sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_x \int_{\mathcal{F}} x^2 y\ dA \\
T_z &= \frac{1}{2}\sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_z \int_{\mathcal{F}} z^2\ dA & T_{yz} &= \frac{1}{2}\sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_y \int_{\mathcal{F}} y^2 z\ dA \\
& & T_{zx} &= \frac{1}{2}\sum_{\mathcal{F}\in\partial\mathcal{V}} \hat{n}_z \int_{\mathcal{F}} z^2 x\ dA.
\end{aligned} \tag{6.3}$$

6.2.2 Reduction to projection integrals

Green's theorem reduces an integral over a planar region to an integral around its one-dimensional boundary; however one must start with a region *in the plane.* Although the planar faces of the polyhedron are in three-space, one can project them onto one of the coordinate planes. The next theorem relates integrations over the original face to integrations over the projection.

Theorem 21 *Let $\mathcal{F}$ be a polygonal region in α-β-γ space, with surface normal $\hat{\mathbf{n}}$, and lying in the plane*

$$\hat{n}_\alpha \alpha + \hat{n}_\beta \beta + \hat{n}_\gamma \gamma + w = 0. \tag{6.4}$$

Let Π be the projection of $\mathcal{F}$ into the α-β plane. Then

$$\int_{\mathcal{F}} f(\alpha, \beta, \gamma)\ dA = \frac{1}{|\hat{n}_\gamma|} \int_{\Pi} f(\alpha, \beta, h(\alpha, \beta))\ d\alpha\ d\beta,$$

where

$$h(\alpha, \beta) = -\frac{1}{\hat{n}_\gamma}(\hat{n}_\alpha \alpha + \hat{n}_\beta \beta + w).$$

Proof: The points $(\alpha, \beta, h(\alpha, \beta))$ lie in the plane of $\mathcal{F}$, so $\mathcal{F}$ is the graph of h over Π. From [EP86] [Section 17.5, Formula (6)],

$$\int_{\mathcal{F}} f(\alpha, \beta, \gamma)\ dA = \int_{\Pi} f(\alpha, \beta, h(\alpha, \beta)) \sqrt{1 + \left(\frac{\partial h}{\partial \alpha}\right)^2 + \left(\frac{\partial h}{\partial \beta}\right)^2}\ d\alpha\ d\beta.$$

The square root in the integrand reduces to $|\hat{n}_\gamma|^{-1}$, and the theorem follows. $\square$

This theorem provides the reduction of the integral of a polynomial in α, β, and γ over the face $\mathcal{F}$ to the integral of a polynomial in α and β over the projected region Π. From (6.4), the constant w can be computed: $w = -\hat{\mathbf{n}} \cdot \mathbf{p}$, where $\mathbf{p}$ is any point on the face $\mathcal{F}$. Numerical inaccuracy or floating point errors can occur when the face normal $\hat{\mathbf{n}}$ has little or no component in the projection direction; in the extreme situation ($\hat{n}_\gamma = 0$), the face projects to a line segment. To reduce such errors, for a given face the α-β-γ coordinates are always chosen as a right-handed[1] permutation of the of the x-y-z coordinates such that $|\hat{n}_\gamma|$ is maximized. This choice maximizes the area of the projected shadow in the α-β plane (see Figure 6.2). Note that a choice can always be found such that $|\hat{n}_\gamma| \geq \sqrt{3}^{-1}$.

Recall the needed integrals (6.3) over the region $\mathcal{F}$. Independent of the three possible correspondences between the x-y-z and α-β-γ coordinates, they all can be found

[1] This means $\hat{\boldsymbol{\alpha}} \times \hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\gamma}}$.

by computing the following 12 integrals:

$$
\begin{array}{llll}
\int_{\mathcal{F}} \alpha \; dA & \int_{\mathcal{F}} \alpha^2 \; dA & \int_{\mathcal{F}} \alpha^3 \; dA & \int_{\mathcal{F}} \alpha^2\beta \; dA \\
\int_{\mathcal{F}} \beta \; dA & \int_{\mathcal{F}} \beta^2 \; dA & \int_{\mathcal{F}} \beta^3 \; dA & \int_{\mathcal{F}} \beta^2\gamma \; dA \\
\int_{\mathcal{F}} \gamma \; dA & \int_{\mathcal{F}} \gamma^2 \; dA & \int_{\mathcal{F}} \gamma^3 \; dA & \int_{\mathcal{F}} \gamma^2\alpha \; dA.
\end{array}
$$

Using Theorem 21, these 12 face integrals can all be reduced to integrals over the projection region Π. For instance,

$$
\begin{aligned}
\int_{\mathcal{F}} \beta^2\gamma \; dA &= |\hat{n}_\gamma|^{-1} \int_{\Pi} \beta^2 \frac{\hat{n}_\alpha \alpha + \hat{n}_\beta \beta + w}{-\hat{n}_\gamma} \; d\alpha \; d\beta \\
&= -|\hat{n}_\gamma|^{-1} \hat{n}_\gamma^{-1} \left(\hat{n}_\alpha \int_{\Pi} \alpha\beta^2 \; d\alpha \; d\beta \; + \; \hat{n}_\beta \int_{\Pi} \beta^3 \; d\alpha \; d\beta \; + \; w \int_{\Pi} \beta^2 \; d\alpha \; d\beta \right) \\
&= -|\hat{n}_\gamma|^{-1} \hat{n}_\gamma^{-1} (\hat{n}_\alpha \pi_{\alpha\beta^2} + \hat{n}_\beta \pi_{\beta^3} + w \pi_{\beta^2}),
\end{aligned}
$$

where

$$
\pi_f = \int_{\Pi} f \; dA. \tag{6.5}
$$

The complete set of face integrals, reduced to projection integrals with Theorem 21, is shown below:

$$\begin{aligned}
\int_{\mathcal{F}} \alpha \; dA &= |\hat{n}_\gamma|^{-1}\pi_\alpha \\
\int_{\mathcal{F}} \beta \; dA &= |\hat{n}_\gamma|^{-1}\pi_\beta \\
\int_{\mathcal{F}} \gamma \; dA &= -|\hat{n}_\gamma|^{-1}\hat{n}_\gamma^{-1}(\hat{n}_\alpha\pi_\alpha + \hat{n}_\beta\pi_\beta + w\pi_1) \\
\int_{\mathcal{F}} \alpha^2 \; dA &= |\hat{n}_\gamma|^{-1}\pi_{\alpha^2} \\
\int_{\mathcal{F}} \beta^2 \; dA &= |\hat{n}_\gamma|^{-1}\pi_{\beta^2} \\
\int_{\mathcal{F}} \gamma^2 \; dA &= |\hat{n}_\gamma|^{-1}\hat{n}_\gamma^{-2}(\hat{n}_\alpha^2\pi_{\alpha^2} + 2\hat{n}_\alpha\hat{n}_\beta\pi_{\alpha\beta} + \hat{n}_\beta^2\pi_{\beta^2} + 2\hat{n}_\alpha w\pi_\alpha + 2\hat{n}_\beta w\pi_\beta + w^2\pi_1) \\
\int_{\mathcal{F}} \alpha^3 \; dA &= |\hat{n}_\gamma|^{-1}\pi_{\alpha^3} \\
\int_{\mathcal{F}} \beta^3 \; dA &= |\hat{n}_\gamma|^{-1}\pi_{\beta^3} \\
\int_{\mathcal{F}} \gamma^3 \; dA &= -|\hat{n}_\gamma|^{-1}\hat{n}_\gamma^{-3}(\hat{n}_\alpha^3\pi_{\alpha^3} + 3\hat{n}_\alpha^2\hat{n}_\beta\pi_{\alpha^2\beta} + 3\hat{n}_\alpha\hat{n}_\beta^2\pi_{\alpha\beta^2} + \hat{n}_\beta^3\pi_{\beta^3} + 3\hat{n}_\alpha^2 w\pi_{\alpha^2} \\
&\quad + 6\hat{n}_\alpha\hat{n}_\beta w\pi_{\alpha\beta} + 3\hat{n}_\beta^2 w\pi_{\beta^2} + 3\hat{n}_\alpha w^2\pi_\alpha + 3\hat{n}_\beta w^2\pi_\beta + w^3\pi_1) \\
\int_{\mathcal{F}} \alpha^2\beta \; dA &= |\hat{n}_\gamma|^{-1}\pi_{\alpha^2\beta} \\
\int_{\mathcal{F}} \beta^2\gamma \; dA &= -|\hat{n}_\gamma|^{-1}\hat{n}_\gamma^{-1}(\hat{n}_\alpha\pi_{\alpha\beta^2} + \hat{n}_\beta\pi_{\beta^3} + w\pi_{\beta^2}) \\
\int_{\mathcal{F}} \gamma^2\alpha \; dA &= |\hat{n}_\gamma|^{-1}\hat{n}_\gamma^{-2}(\hat{n}_\alpha^2\pi_{\alpha^3} + 2\hat{n}_\alpha\hat{n}_\beta\pi_{\alpha^2\beta} + \hat{n}_\beta^2\pi_{\alpha\beta^2} + 2\hat{n}_\alpha w\pi_{\alpha^2} + 2\hat{n}_\beta w\pi_{\alpha\beta} + w^2\pi_\alpha).
\end{aligned} \tag{6.6}$$

6.2.3 Reduction to line integrals

The final step is to reduce an integral over a polygonal projection region in the α-β plane to a sum of line integrals over the edges bounding the region. The following notation is used (see Figure 6.3). The edges of Π are labeled $\mathcal{E}_1$ through $\mathcal{E}_K$. Edge $\mathcal{E}_e$ is the directed line segment from (α_e, β_e) to $(\alpha_{e+1}, \beta_{e+1})$, $\Delta\alpha_e = \alpha_{e+1} - \alpha_e$, and $\Delta\beta_e = \beta_{e+1} - \beta_e$ [note that $(\alpha_{K+1}, \beta_{K+1}) = (\alpha_1, \beta_1)$]. Finally, edge $\mathcal{E}_e$ has length L_e and exterior unit normal $\hat{\mathbf{m}}_e$. Green's theorem[2] [Ste82] provides the final integration reduction:

[2]Sometimes more formally called *Green's theorem in the plane*. Additionally, some texts call this *Green's lemma*, reserving *Green's theorem* for a more general 3D result [WB82].

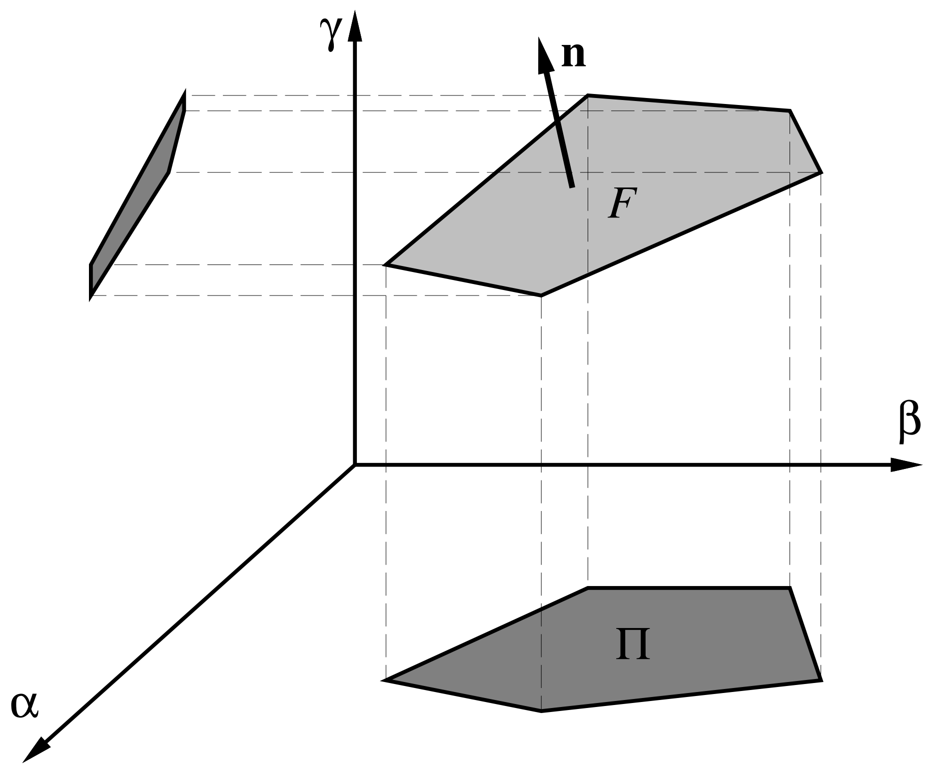

Figure 6.2: *The α-β-γ axes are a right-handed permutation of the x-y-z axes chosen to maximize the size of the face's projected shadow in the α-β plane.*

Theorem 22 (Green's) *Let Π be a region in the plane bounded by the single curve $\partial\Pi$. Let $\hat{\mathbf{m}}$ denote the exterior normal along the boundary. Then*

$$\int_{\Pi} \boldsymbol{\nabla} \cdot \mathbf{H}\; dA = \oint_{\partial\Pi} \mathbf{H} \cdot \hat{\mathbf{m}}\; ds \tag{6.7}$$

for any vector field $\mathbf{H}$ defined on Π, where the line integral traverses the boundary counter-clockwise.

This theorem is a two-dimensional version of the divergence theorem, and the polyhedral application again provides simplification. Since Π is polygonal, the right-hand side of (6.7) may be broken into a summation over edges of constant normal, and by always choosing $\mathbf{H}$ so that one component is identically zero, the dot product becomes a scalar multiplication. From (6.5) and (6.6), all integrals of the form

$$\pi_{\alpha^p \beta^q} = \int_{\Pi} \alpha^p \beta^q \; dA$$

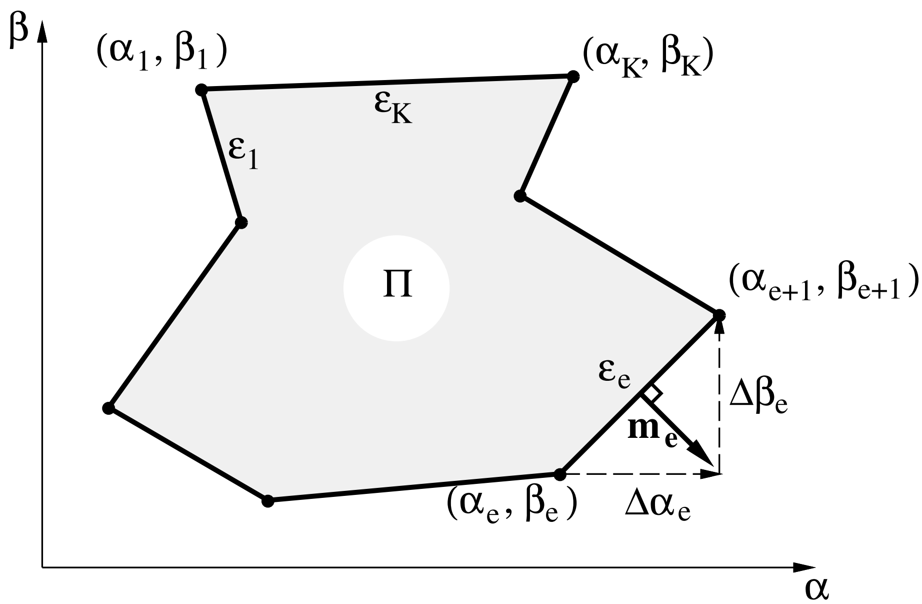

Figure 6.3: *Notation for computing a projection integral as a sum of line integrals.*

are needed for nonnegative integers p and q with $p+q \leq 3$. Consider first the case $q=0$. By choosing $\mathbf{H} = (\frac{1}{p+1}\alpha^{p+1}, 0)^T$, and applying Green's theorem to the polygonal region Π,

$$\int_\Pi \alpha^p \ dA = \frac{1}{p+1} \sum_{e=1}^{K} \hat{m}_{e_\alpha} \int_{\mathcal{E}_e} \alpha(s)^{p+1} \ ds.$$

In the right-hand integral, the integration variable s is arc length, and runs from 0 to L_e, the length of the edge; $\alpha(s)$ is the α-coordinate of the point on the edge that is a distance s from the starting point. Letting the variable λ be s/L_e, one can change integration variables ($ds = L_e d\lambda$) to get

$$\int_\Pi \alpha^p \ dA = \frac{1}{p+1} \sum_{e=1}^{K} \hat{m}_{e_\alpha} L_e \int_0^1 \alpha(L_e\lambda)^{p+1} \ d\lambda.$$

Now $\hat{m}_{e_\alpha} L_e = \pm\Delta\beta_e$, where the plus or minus depends on whether the vertices of Π are indexed in counterclockwise or clockwise order, respectively. By convention, assume that the vertices of the original face $\mathcal{F}$ are indexed in counterclockwise order, thus the vertices of Π will be indexed in counterclockwise order exactly when the sign of $\hat{n}_\gamma$ is positive (Figure 6.2 helps in visualizing this situation). Hence, $\hat{m}_{e_\alpha} L_e = \operatorname{sgn}(\hat{n}_\gamma)\Delta\beta_e$, and

$$\int_\Pi \alpha^p \ dA = \frac{\operatorname{sgn}(\hat{n}_\gamma)}{p+1} \sum_{e=1}^{K} \Delta\beta_e \int_0^1 \alpha(L_e\lambda)^{p+1} \ d\lambda. \tag{6.8}$$

The case with $q = 1$ is similar, except one chooses $\mathbf{H} = (\frac{1}{p+1}\alpha^{p+1}\beta, 0)^T$. Finally, one can derive analogous equations for the cases when $p = 0$ or $p = 1$. The results are:

$$\begin{aligned}
\int_\Pi \beta^q \, dA &= -\frac{\operatorname{sgn}(\hat{n}_\gamma)}{q+1} \sum_{e=1}^{K} \Delta\alpha_e \int_0^1 \beta(L_e\lambda)^{q+1} \, d\lambda \\
\int_\Pi \alpha^p \beta \, dA &= \frac{\operatorname{sgn}(\hat{n}_\gamma)}{p+1} \sum_{e=1}^{K} \Delta\beta_e \int_0^1 \alpha(L_e\lambda)^{p+1} \beta(L_e\lambda) \, d\lambda \\
\int_\Pi \alpha\beta^q \, dA &= -\frac{\operatorname{sgn}(\hat{n}_\gamma)}{q+1} \sum_{e=1}^{K} \Delta\alpha_e \int_0^1 \alpha(L_e\lambda)\beta(L_e\lambda)^{q+1} \, d\lambda.
\end{aligned} \tag{6.9}$$

6.2.4 Evaluation of integrals from vertex coordinates

The original volume integrals have been successively reduced to face integrals, projection integrals, and finally line integrals. The latter can be directly evaluated in terms of vertex coordinates, with the help of the following theorem.

Theorem 23 *Let L_e be the length of the directed line segment from (α_e, β_e) to $(\alpha_{e+1}, \beta_{e+1})$. Let $\alpha(s)$ and $\beta(s)$ be the α- and β-coordinates of the point on this segment a distance s from the starting point. Then for nonnegative integers p and q,*

$$\int_0^1 \alpha(L_e\lambda)^p \beta(L_e\lambda)^q \, d\lambda = \frac{1}{p+q+1} \sum_{i=0}^{p} \sum_{j=0}^{q} \frac{\binom{p}{i}\binom{q}{j}}{\binom{p+q}{i+j}} \alpha_{e+1}^i \alpha_e^{p-i} \beta_{e+1}^j \beta_e^{q-j}.$$

Proof: Denoting the integral on the left-hand side by I,

$$\begin{aligned}
I &= \int_0^1 [(1-\lambda)\alpha_e + \lambda\alpha_{e+1}]^p \, [(1-\lambda)\beta_e + \lambda\beta_{e+1}]^q \, d\lambda \\
&= \int_0^1 \left[\sum_{i=0}^{p} B_i^p(\lambda)\alpha_{e+1}^i \alpha_e^{p-i}\right] \left[\sum_{j=0}^{q} B_j^q(\lambda)\beta_{e+1}^j \beta_e^{q-j}\right] \, d\lambda,
\end{aligned} \tag{6.10}$$

where the B's are *Bernstein polynomials*:

$$B_k^n(\lambda) = \binom{n}{k} \lambda^k (1-\lambda)^{n-k}.$$

Two important properties of these polynomials are [FR88, Far90]:

$$\begin{aligned} B_i^p(\lambda)B_j^q(\lambda) &= \frac{\begin{pmatrix} p \\ i \end{pmatrix}\begin{pmatrix} q \\ j \end{pmatrix}}{\begin{pmatrix} p+q \\ i+j \end{pmatrix}} B_{i+j}^{p+q}(\lambda), & (6.11) \\ \int_0^1 B_k^n(\lambda)\ d\lambda &= \frac{1}{n+1}. & (6.12) \end{aligned}$$

Expanding the product in (6.10) and applying (6.11) gives

$$I = \sum_{i=0}^{p}\sum_{j=0}^{q} \frac{\begin{pmatrix} p \\ i \end{pmatrix}\begin{pmatrix} q \\ j \end{pmatrix}}{\begin{pmatrix} p+q \\ i+j \end{pmatrix}} \alpha_{e+1}^i \alpha_e^{p-i} \beta_{e+1}^j \beta_e^{q-j} \int_0^1 B_{i+j}^{p+q}(\lambda)\ d\lambda. \tag{6.13}$$

Evaluating the integrals using (6.12) proves the theorem. □

All of the line integrals appearing in (6.8–6.9) are special cases of Theorem 23, with either p or q set to 0 or 1. Specifically,

$$\begin{aligned} \int_0^1 \alpha^{p+1}\ d\lambda &= \frac{1}{p+2}\sum_{i=0}^{p+1} \alpha_{e+1}^i \alpha_e^{p+1-i} \\ \int_0^1 \beta^{q+1}\ d\lambda &= \frac{1}{q+2}\sum_{j=0}^{q+1} \beta_{e+1}^j \beta_e^{q+1-j} \\ \int_0^1 \alpha^{p+1}\beta\ d\lambda &= \frac{1}{(p+2)(p+3)}\left[\beta_{e+1}\sum_{i=0}^{p+1}(i+1)\alpha_{e+1}^i\alpha_e^{p+1-i} + \beta_e\sum_{i=0}^{p+1}(p+2-i)\alpha_{e+1}^i\alpha_e^{p+1-i}\right] \\ \int_0^1 \alpha\beta^{q+1}\ d\lambda &= \frac{1}{(q+2)(q+3)}\left[\alpha_{e+1}\sum_{j=0}^{q+1}(j+1)\beta_{e+1}^j\beta_e^{q+1-j} + \alpha_e\sum_{j=0}^{q+1}(q+2-j)\beta_{e+1}^j\beta_e^{q+1-j}\right]. \end{aligned}$$

Making these substitutions into (6.8)–(6.9) give all of the needed projection integrals in terms of the coordinates of the projection vertices:

$$
\begin{aligned}
\pi_1 = \int_\Pi 1 \; dA \quad &= \quad +\frac{\operatorname{sgn}(\hat{n}_\gamma)}{2} \sum_{e=1}^{K} \Delta\beta_e(\alpha_{e+1} + \alpha_e) \\
\pi_\alpha = \int_\Pi \alpha \; dA \quad &= \quad +\frac{\operatorname{sgn}(\hat{n}_\gamma)}{6} \sum_{e=1}^{K} \Delta\beta_e(\alpha_{e+1}^2 + \alpha_{e+1}\alpha_e + \alpha_e^2) \\
\pi_\beta = \int_\Pi \beta \; dA \quad &= \quad -\frac{\operatorname{sgn}(\hat{n}_\gamma)}{6} \sum_{e=1}^{K} \Delta\alpha_e(\beta_{e+1}^2 + \beta_{e+1}\beta_e + \beta_e^2) \\
\pi_{\alpha^2} = \int_\Pi \alpha^2 \; dA \quad &= \quad +\frac{\operatorname{sgn}(\hat{n}_\gamma)}{12} \sum_{e=1}^{K} \Delta\beta_e(\alpha_{e+1}^3 + \alpha_{e+1}^2 a_e + \alpha_{e+1}\alpha_e^2 + \alpha_e^3) \\
\pi_{\alpha\beta} = \int_\Pi \alpha\beta \; dA \quad &= \quad +\frac{\operatorname{sgn}(\hat{n}_\gamma)}{24} \sum_{e=1}^{K} \Delta\beta_e \Big[\beta_{e+1}(3\alpha_{e+1}^2 + 2\alpha_{e+1}\alpha_e + \alpha_e^2) \\
&\qquad\qquad\qquad + \beta_e(\alpha_{e+1}^2 + 2\alpha_e\alpha_{e+1} + 3\alpha_e^2)\Big] \\
\pi_{\beta^2} = \int_\Pi \beta^2 \; dA \quad &= \quad -\frac{\operatorname{sgn}(\hat{n}_\gamma)}{12} \sum_{e=1}^{K} \Delta\alpha_e(\beta_{e+1}^3 + \beta_{e+1}^2\beta_e + \beta_{e+1}\beta_e^2 + \beta_e^3) \\
\pi_{\alpha^3} = \int_\Pi \alpha^3 \; dA \quad &= \quad +\frac{\operatorname{sgn}(\hat{n}_\gamma)}{20} \sum_{e=1}^{K} \Delta\beta_e(\alpha_{e+1}^4 + \alpha_{e+1}^3\alpha_e + \alpha_{e+1}^2\alpha_e^2 + \alpha_{e+1}\alpha_e^3 + \alpha_e^4) \\
\pi_{\alpha^2\beta} = \int_\Pi \alpha^2\beta \; dA \quad &= \quad +\frac{\operatorname{sgn}(\hat{n}_\gamma)}{60} \sum_{e=1}^{K} \Delta\beta_e \Big[\beta_{e+1}(4\alpha_{e+1}^3 + 3\alpha_{e+1}^2\alpha_e + 2\alpha_{e+1}\alpha_e^2 + \alpha_e^3) \\
&\qquad\qquad\qquad + \beta_e(\alpha_{e+1}^3 + 2\alpha_{e+1}^2\alpha_e + 3\alpha_{e+1}\alpha_e^2 + 4\alpha_e^3)\Big] \\
\pi_{\alpha\beta^2} = \int_\Pi \alpha\beta^2 \; dA \quad &= \quad -\frac{\operatorname{sgn}(\hat{n}_\gamma)}{60} \sum_{e=1}^{K} \Delta\alpha_e \Big[\alpha_{e+1}(4\beta_{e+1}^3 + 3\beta_{e+1}^2\beta_e + 2\beta_{e+1}\beta_e^2 + \beta_e^3) \\
&\qquad\qquad\qquad + \alpha_e(\beta_{e+1}^3 + 2\beta_{e+1}^2\beta_e + 3\beta_{e+1}\beta_e^2 + 4\beta_e^3)\Big] \\
\pi_{\beta^3} = \int_\Pi \beta^3 \; dA \quad &= \quad -\frac{\operatorname{sgn}(\hat{n}_\gamma)}{20} \sum_{e=1}^{K} \Delta\alpha_e(\beta_{e+1}^4 + \beta_{e+1}^3\beta_e + \beta_{e+1}^2\beta_e^2 + \beta_{e+1}\beta_e^3 + \beta_e^4).
\end{aligned}
\tag{6.14}
$$

6.3 Pseudocode and on-line C code

The derivations of the previous section specify a complete algorithm for computing the ten desired volume integrals (6.1). The algorithm comprises three routines:

1. `CompVolumeIntegrals(`$\mathcal{V}$`)` (Figure 6.4) computes the required volume integrals for a polyhedron by summing surface integrals over its faces, as detailed in Equations (6.3).
2. `CompFaceIntegrals(`$\mathcal{F}$`)` (Figure 6.5) computes the required surface integrals over a polyhedral face from the integrals over its projection, as detailed in Equations (6.6).
3. `CompProjectionIntegrals(`$\mathcal{F}$`)` (Figure 6.6) computes the required integrals over a face projection from the coordinates of the projections vertices, as detailed in Equations (6.14).

This algorithm contains a minor simplification over the presented derivation. When Equations (6.14) are substituted into (6.6), the computation of $\text{sgn}(\hat{n}_\gamma)$ and $|\hat{n}_\gamma|$ becomes unnecessary, since these factors always appear together in a product, giving simply $\hat{n}_\gamma$. Thus, no signs or absolute values are computed in the routines `CompFaceIntegrals` and `CompProjectionIntegrals`.

ANSI C source code for the algorithm described in this paper, and detailed in Figures 6.4–6.6, is publicly available from two web sites:

`http://www.acm.org/jgt,`

`http://www.cs.berkeley.edu/~mirtich/massProps.html`

Also included is an interface to build up polyhedra, using a simple data structure, from ASCII specifications; examples are provided. The code is public domain, and may be used as is or in modified form.

6.4 Test results

This section analyzes the speed and accuracy of the algorithm for various test cases. These tests were run on an *SGI Indigo II* with a 200 MHz R4400 CPU, and double precision floating point numbers were used for the calculations.

The set of polyhedral objects that have volume integrals which are commonly tabulated or easy to compute by hand is rather limited. The algorithm was run on two such objects: an axes-aligned cube, 20 units on a side and centered at the origin; and a tetrahedron defined by the convex hull of the origin, and the vectors $5\hat{\mathbf{i}}$, $4\hat{\mathbf{j}}$, and $3\hat{\mathbf{k}}$. The theoretical values for these objects are shown in Table 6.1. For these two examples, all

object	T_1	T_x	T_y	T_z	T_{x^2}	T_{y^2}	T_{z^2}	T_{xy}	T_{yz}	T_{zx}
cube	8000	0	0	0	2.67×10^5	2.67×10^5	2.67×10^5	0	0	0
tetrahedron	10	12.5	10	7.5	25	16	9	10	6	7.5

Table 6.1: Theoretical values of volume integrals for simple test polyhedra.

values computed by the algorithm were correct to at least 15 significant figures. The total time required to compute all ten volume integrals was 64 μs for the tetrahedron, and 110 μs for the cube.

For a more interesting test, the algorithm was applied to several polyhedral approximations of a unit radius sphere, centered at the origin. In this case there are two sources of error: numerical errors from the algorithm, and approximation errors inherent in the geometric model, which is not a true sphere. These latter errors should not be attributed to the algorithm itself. For a perfect unit sphere, the integrals $T_x, T_y, T_z, T_{xy}, T_{yz}$, and T_{zx} should vanish, while $T_1 = \frac{4}{3}\pi$ and $T_{x^2} = T_{y^2} = T_{z^2} = \frac{4}{15}\pi$. The algorithm was applied to a series of successive approximations to the sphere, beginning with an icosahedron, and obtaining each finer approximation by projecting the midpoint of each polyhedral edge onto the unit sphere, and taking a convex hull. The computed values of a representative set of volume integrals for each polyhedron are shown in Table 6.2.

Without numerical error, the integrals T_x and T_{yz} would vanish for all six polyhedral approximations of the sphere, due to symmetry. From Table 6.2, the absolute values of these computed values are all less than 10^{-15}. The theoretical values in the table correspond to the sphere which circumscribes the polyhedra. For each polyhedron, the corresponding values for the inscribed sphere were also determined, and the computed values for T_1 and T_{x^2} verified to lie between the bounding values for these two spheres. For approximation 6, the difference in values for the inscribed and circumscribed sphere is 2.8×10^{-3} for T_1 and 9.5×10^{-4} for T_{x^2}. These values are upper bounds on the numerical errors of the algorithm. Note that the deviations between theoretical and computed values for T_1 and T_{x^2} are reduced as the complexity of the polyhedron increases, while numerical error from the algorithm should increase with complexity. In light of the very small errors incurred in the computation of T_x and T_{yz}, the deviations between the computed and theoretical values of T_1 and T_{x^2} are almost certainly due mainly to the polyhedral approximation rather than to numerical errors.

approx.	verts	edges	faces	T_1	T_x	T_{x^2}	T_{yz}	time
1	12	30	20	2.536	-2.8×10^{-17}	0.3670	-3.1×10^{-17}	500 μs
2	42	120	80	3.659	$+1.4 \times 10^{-16}$	0.6692	$+1.5 \times 10^{-17}$	1.2 ms
3	162	480	320	4.047	-3.2×10^{-16}	0.7911	-6.1×10^{-18}	4.9 ms
4	642	1920	1280	4.153	$+3.0 \times 10^{-16}$	0.8258	$+7.8 \times 10^{-18}$	21 ms
5	2562	7680	5120	4.180	-3.8×10^{-17}	0.8347	$+2.1 \times 10^{-17}$	82 ms
6	10242	30720	20480	4.187	$+5.6 \times 10^{-16}$	0.8370	$+6.4 \times 10^{-18}$	350 ms
theoretical values for sphere				4.189	0.0	0.8378	0.0	–

Table 6.2: Data for successive approximations of a unit sphere.

The execution times shown in Table 6.2 are the total times for computing all ten volume integrals for each polyhedron. The $O(n)$ nature of the algorithm is evident: from approximation 2 on, the time ratios between successive refinements very closely follow the 4:1 ratio in the number of faces. The algorithm is also very fast: all ten integrals are computed for a polyhedron with over 20,000 faces in only 350 ms.

Acknowledgments. This chapter originally appeared as the paper [Mir96]. We thank Aristides Requicha for a valuable literature survey on this topic, and David Baraff for useful comments on the initial draft of this paper. We especially thank John Hughes for his detailed review and many suggestions for improving the paper.

```
CompVolumeIntegrals(𝒱)

    T_1, T_x, T_y, T_z, T_{x^2}, T_{y^2}, T_{z^2}, T_{xy}, T_{yz}, T_{zx} ← 0

    for each face ℱ on the boundary of 𝒱
        (α, β, γ) ← right-handed permutation of (x, y, z) that maximizes |n̂_γ|
        compFaceIntegrals(ℱ)
        if (α = x) T_1 ← T_1 + n̂_α F_α
        else if (β = x) T_1 ← T_1 + n̂_β F_β
        else T_1 ← T_1 + n̂_γ F_γ
        T_α ← T_α + n̂_α F_{α^2}
        T_β ← T_β + n̂_β F_{β^2}
        T_γ ← T_γ + n̂_γ F_{γ^2}
        T_{α^2} ← T_{α^2} + n̂_α F_{α^3}
        T_{β^2} ← T_{β^2} + n̂_β F_{β^3}
        T_{γ^2} ← T_{γ^2} + n̂_γ F_{γ^3}
        T_{αβ} ← T_{αβ} + n̂_α F_{α^2 β}
        T_{βγ} ← T_{βγ} + n̂_β F_{β^2 γ}
        T_{γα} ← T_{γα} + n̂_γ F_{γ^2 α}

    (T_x, T_y, T_z) ← (T_x, T_y, T_z) / 2
    (T_{x^2}, T_{y^2}, T_{z^2}) ← (T_{x^2}, T_{y^2}, T_{z^2}) / 3
    (T_{xy}, T_{yz}, T_{zx}) ← (T_{xy}, T_{yz}, T_{zx}) / 2
```

Figure 6.4: `CompVolumeIntegrals(`$\mathcal{V}$`)`. Compute the required volume integrals for a polyhedron. See Equations (6.3).

```
CompFaceIntegrals(𝓕)

    computeProjectionIntegrals(𝓕)
```

$w \leftarrow -\hat{\mathbf{n}} \cdot \mathbf{p}$ `for some point` $\mathbf{p}$ `on` $\mathcal{F}$

$k_1 \leftarrow \hat{n}_\gamma^{-1}$; $k_2 \leftarrow k_1 * k_1$; $k_3 \leftarrow k_2 * k_1$; $k_4 \leftarrow k_3 * k_1$

$$F_\alpha \leftarrow k_1 \pi_\alpha$$
$$F_\beta \leftarrow k_1 \pi_\beta$$
$$F_\gamma \leftarrow -k_2(\hat{n}_\alpha \pi_\alpha + \hat{n}_\beta \pi_\beta + w\pi_1)$$

$$F_{\alpha^2} \leftarrow k_1 \pi_{\alpha^2}$$
$$F_{\beta^2} \leftarrow k_1 \pi_{\beta^2}$$
$$F_{\gamma^2} \leftarrow k_3(\hat{n}_\alpha^2 \pi_{\alpha^2} + 2\hat{n}_\alpha \hat{n}_\beta \pi_{\alpha\beta} + \hat{n}_\beta^2 \pi_{\beta^2} + 2\hat{n}_\alpha w \pi_\alpha + 2\hat{n}_\beta w \pi_\beta + w^2 \pi_1)$$

$$F_{\alpha^3} \leftarrow k_1 \pi_{\alpha^3}$$
$$F_{\beta^3} \leftarrow k_1 \pi_{\beta^3}$$
$$\begin{aligned} F_{\gamma^3} \leftarrow -k_4(&\hat{n}_\alpha^3 \pi_{\alpha^3} + 3\hat{n}_\alpha^2 \hat{n}_\beta \pi_{\alpha^2\beta} + 3\hat{n}_\alpha \hat{n}_\beta^2 \pi_{\alpha\beta^2} + \hat{n}_\beta^3 \pi_{\beta^3} \\ &+ 3\hat{n}_\alpha^2 w \pi_{\alpha^2} + 6\hat{n}_\alpha \hat{n}_\beta w \pi_{\alpha\beta} + 3\hat{n}_\beta^2 w \pi_{\beta^2} + 3\hat{n}_\alpha w^2 \pi_\alpha + 3\hat{n}_\beta w^2 \pi_\beta + w^3 \pi_1) \end{aligned}$$

$$F_{\alpha^2\beta} \leftarrow k_1 \pi_{\alpha^2\beta}$$
$$F_{\beta^2\gamma} \leftarrow -k_2(\hat{n}_\alpha \pi_{\alpha\beta^2} + \hat{n}_\beta \pi_{\beta^3} + w\pi_{\beta^2})$$
$$F_{\gamma^2\alpha} \leftarrow k_3(\hat{n}_\alpha^2 \pi_{\alpha^3} + 2\hat{n}_\alpha \hat{n}_\beta \pi_{\alpha^2\beta} + \hat{n}_\beta^2 \pi_{\alpha\beta^2} + 2\hat{n}_\alpha w \pi_{\alpha^2} + 2\hat{n}_\beta w \pi_{\alpha\beta} + w^2 \pi_\alpha)$$

Figure 6.5: `CompFaceIntegrals(𝓕)`. Compute the required surface integrals over a polyhedral face. See Equations (6.6).

```
CompProjectionIntegrals(𝓕)
```

$\pi_1, \pi_\alpha, \pi_b, \pi_{\alpha^2}, \pi_{\alpha\beta}, \pi_{\beta^2}, \pi_{\alpha^3}, \pi_{\alpha^2\beta}, \pi_{\alpha\beta^2}, \pi_{\beta^3} \leftarrow 0$

for each edge $\mathcal{E}$ in CCW order around $\mathcal{F}$

$\alpha_0 \leftarrow \alpha$-coordinate of start point of $\mathcal{E}$
$\beta_0 \leftarrow \beta$-coordinate of start point of $\mathcal{E}$
$\alpha_1 \leftarrow \alpha$-coordinate of end point of $\mathcal{E}$
$\beta_1 \leftarrow \beta$-coordinate of end point of $\mathcal{E}$
$\Delta\alpha \leftarrow \alpha_1 - \alpha_0$
$\Delta\beta \leftarrow \beta_1 - \beta_0$
$\alpha_0^2 \leftarrow \alpha_0 * \alpha_0$; $\alpha_0^3 \leftarrow \alpha_0^2 * \alpha_0$; $\alpha_0^4 \leftarrow \alpha_0^3 * \alpha_0$
$\beta_0^2 \leftarrow \beta_0 * \beta_0$; $\beta_0^3 \leftarrow \beta_0^2 * \beta_0$; $\beta_0^4 \leftarrow \beta_0^3 * \beta_0$
$\alpha_1^2 \leftarrow \alpha_1 * \alpha_1$; $\alpha_1^3 \leftarrow \alpha_1^2 * \alpha_1$
$\beta_1^2 \leftarrow \beta_1 * \beta_1$; $\beta_1^3 \leftarrow \beta_1^2 * \beta_1$

$C_1 \leftarrow \alpha_1 + \alpha_0$
$C_\alpha \leftarrow \alpha_1 C_1 + \alpha_0^2$; $C_{\alpha^2} \leftarrow \alpha_1 C_\alpha + \alpha_0^3$; $C_{\alpha^3} \leftarrow \alpha_1 C_{\alpha^2} + \alpha_0^4$
$C_\beta \leftarrow \beta_1^2 + \beta_1\beta_0 + \beta_0^2$; $C_{\beta^2} \leftarrow \beta_1 C_\beta + \beta_0^3$; $C_{\beta^3} \leftarrow \beta_1 C_{\beta^2} + \beta_0^4$

$C_{\alpha\beta} \leftarrow 3\alpha_1^2 + 2a_1a_0 + a_0^2$; $K_{\alpha\beta} \leftarrow \alpha_1^2 + 2\alpha_1\alpha_0 + 3\alpha_0^2$
$C_{\alpha^2\beta} \leftarrow \alpha_0 C_{\alpha\beta} + 4\alpha_1^3$; $K_{\alpha^2\beta} \leftarrow \alpha_1 K_{\alpha\beta} + 4\alpha_0^3$
$C_{\alpha\beta^2} \leftarrow 4\beta_1^3 + 3\beta_1^2\beta_0 + 2\beta_1\beta_0^2 + \beta_0^3$; $K_{\alpha\beta^2} \leftarrow \beta_1^3 + 2\beta_1^2\beta_0 + 3\beta_1\beta_0^2 + 4\beta_0^3$

$\pi_1 \leftarrow \pi_1 + \Delta\beta C_1$
$\pi_\alpha \leftarrow \pi_\alpha + \Delta\beta C_\alpha$
$\pi_{\alpha^2} \leftarrow \pi_{\alpha^2} + \Delta\beta C_{\alpha^2}$
$\pi_{\alpha^3} \leftarrow \pi_{\alpha^3} + \Delta\beta C_{\alpha^3}$
$\pi_\beta \leftarrow \pi_\beta + \Delta\alpha C_\beta$
$\pi_{\beta^2} \leftarrow \pi_{\beta^2} + \Delta\alpha C_{\beta^2}$
$\pi_{\beta^3} \leftarrow \pi_{\beta^3} + \Delta\alpha C_{\beta^3}$
$\pi_{\alpha\beta} \leftarrow \pi_{\alpha\beta} + \Delta\beta(\beta_1 C_{\alpha\beta} + \beta_0 K_{\alpha\beta})$
$\pi_{\alpha^2\beta} \leftarrow \pi_{\alpha^2\beta} + \Delta\beta(\beta_1 C_{\alpha^2\beta} + \beta_0 K_{\alpha^2\beta})$
$\pi_{\alpha\beta^2} \leftarrow \pi_{\alpha\beta^2} + \Delta\alpha(\alpha_1 C_{\alpha\beta^2} + \alpha_0 K_{\alpha\beta^2})$

$\pi_1 \leftarrow \pi_1/2$
$\pi_\alpha \leftarrow \pi_\alpha/6$
$\pi_{\alpha^2} \leftarrow \pi_{\alpha^2}/12$
$\pi_{\alpha^3} \leftarrow \pi_{\alpha^3}/20$
$\pi_\beta \leftarrow -\pi_\beta/6$
$\pi_{\beta^2} \leftarrow -\pi_{\beta^2}/12$
$\pi_{\beta^3} \leftarrow -\pi_{\beta^3}/20$
$\pi_{\alpha\beta} \leftarrow \pi_{\alpha\beta}/24$
$\pi_{\alpha^2\beta} \leftarrow \pi_{\alpha^2\beta}/60$
$\pi_{\alpha\beta^2} \leftarrow -\pi_{\alpha\beta^2}/60$

Figure 6.6: `CompProjectionIntegrals(𝓕)`. Compute the required integrals over a face projection. See Equations (6.14).

Chapter 7

Examples and Results

The ideas and theory presented in this thesis have been put into practice through the prototype, *Impulse*, an experimental impulse-based dynamic simulator. *Impulse* is written in ANSI C and runs on SGI and HP platforms. This chapter describes several examples of simulations produced using *Impulse*, which fall into three categories: pure impulse-based simulation, passive hybrid simulation, and controlled hybrid simulation. Snapshots of the simulations are included, and most of the simulations described here can be viewed as MPEG movies available from the *Impulse* web site:

http://www.cs.berkeley.edu/~mirtich/impulse.html

Many of these simulations were designed to test specific aspects of the impulse-based approach. Beyond these test cases, *Impulse* has been used as an engineering tool for real applications in the parts feeding domain. These applications and some results are described here. *All of the simulations described here were run using the identical algorithms with absolutely no tuning of parameters.*

7.1 Pure impulse-based simulation

Pure impulse-based simulation means that all contact interaction is handled through trains of collision impulses; there are no explicit constraints governing the relative motion of different bodies. All bodies in the simulation are either fixed in space, or execute piecewise ballistic trajectories between collisions. This basic paradigm adeptly handles colliding and transient contact, and can also model many types of continuous contact, including sliding,

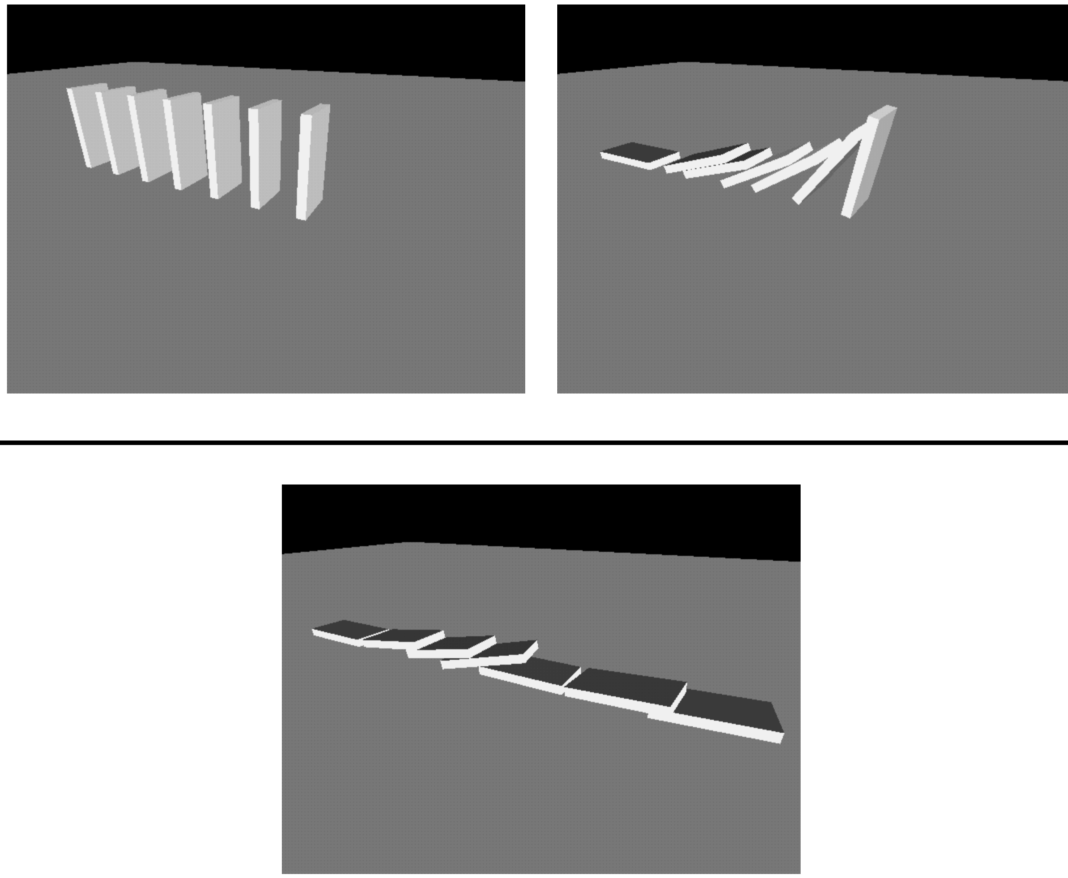

Figure 7.1: *Snapshots from the* dominos *simulation.*

rolling, and resting contact.

Dominos

The *dominos* simulation involves a row of seven dominos that fall over (Figure 7.1). While the dominos are falling, there are many simultaneous contacts between them, which are modeled as individual collisions. The effects of the friction are evident in the simulation: after all dominos have fallen, they come to rest on the supporting surface. Modeling friction is critical to realism, as can be seen by comparing this simulation movie to a simulation

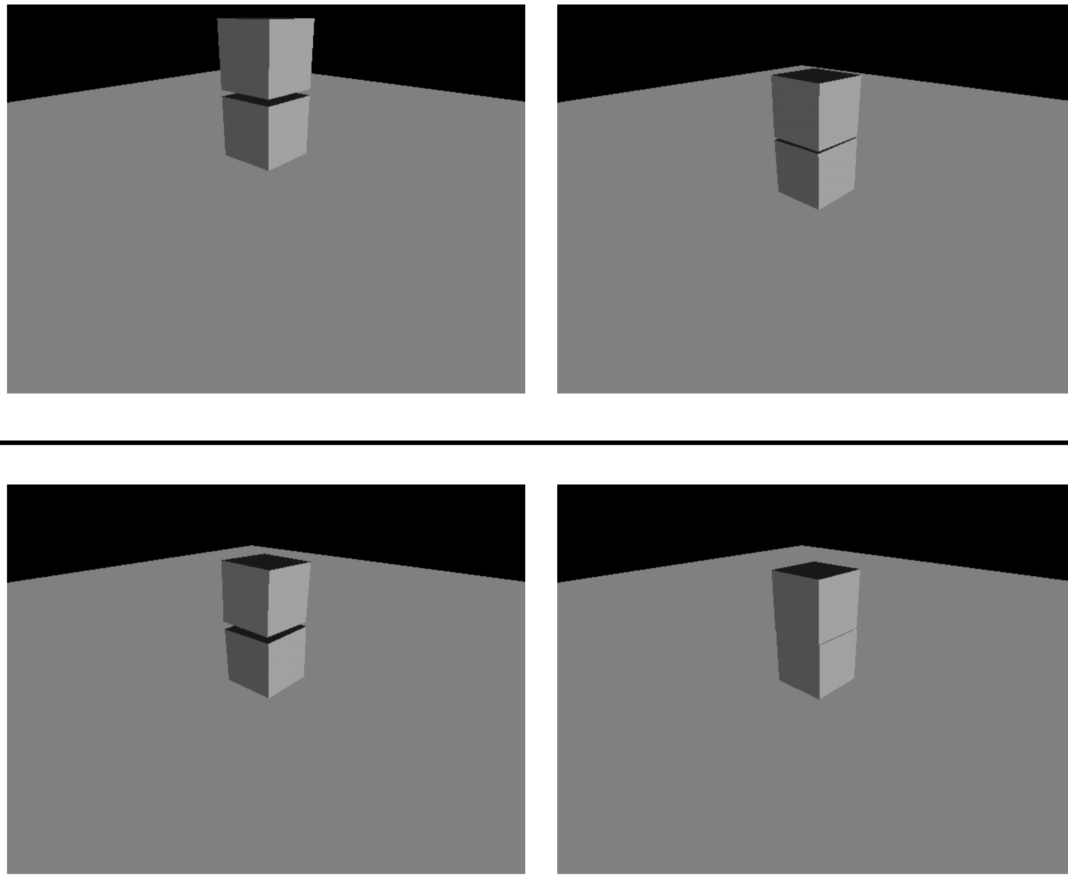

Figure 7.2: *Snapshots from the* block drop *simulation.*

movie of frictionless dominos produced by the *Simpact* system [RK].

Block drop

The *block drop* simulation (Figure 7.2) involves dropping two blocks onto a supporting surface so that the blocks come to rest in a stack. The blocks visibly bounce several times before settling into a resting state modeled with microcollisions. Performing this test with more than two blocks proved troublesome, since collision impulses must be propagated up and down the stack in order to keep blocks from penetrating. Stacks of objects subject to

Figure 7.3: *Snapshots from the* block on ramp *simulation.*

gravitational forces are one type of system poorly modeled with impulse-based simulation: the objects penetrate deeply into collision envelopes, and the collision detection and collision response computations become excessive. Impulse-based simulation can handle large stacks which are in the process of settling, but once the system reaches a static state, it is easier to model with constraints.

Block on ramp

The *block on ramp* simulation involves a block that slides down a ramp, under conditions where the friction between the block and ramp is sufficient to stop the sliding (Figure 7.3). The ramp is at a $30°$ angle with the horizontal, the block is launched with a velocity along the ramp of 548 cm/s, and the coefficient of friction is 0.9. This experiment tests the accuracy of the friction model, since the block initially experiences sliding frictional forces, and later experiences static friction forces when it comes to rest. Under *Impulse*, the block reaches a resting state at very nearly the time and place predicted by theory. Graphs of the block's position and velocity are shown in Figure 3.15.

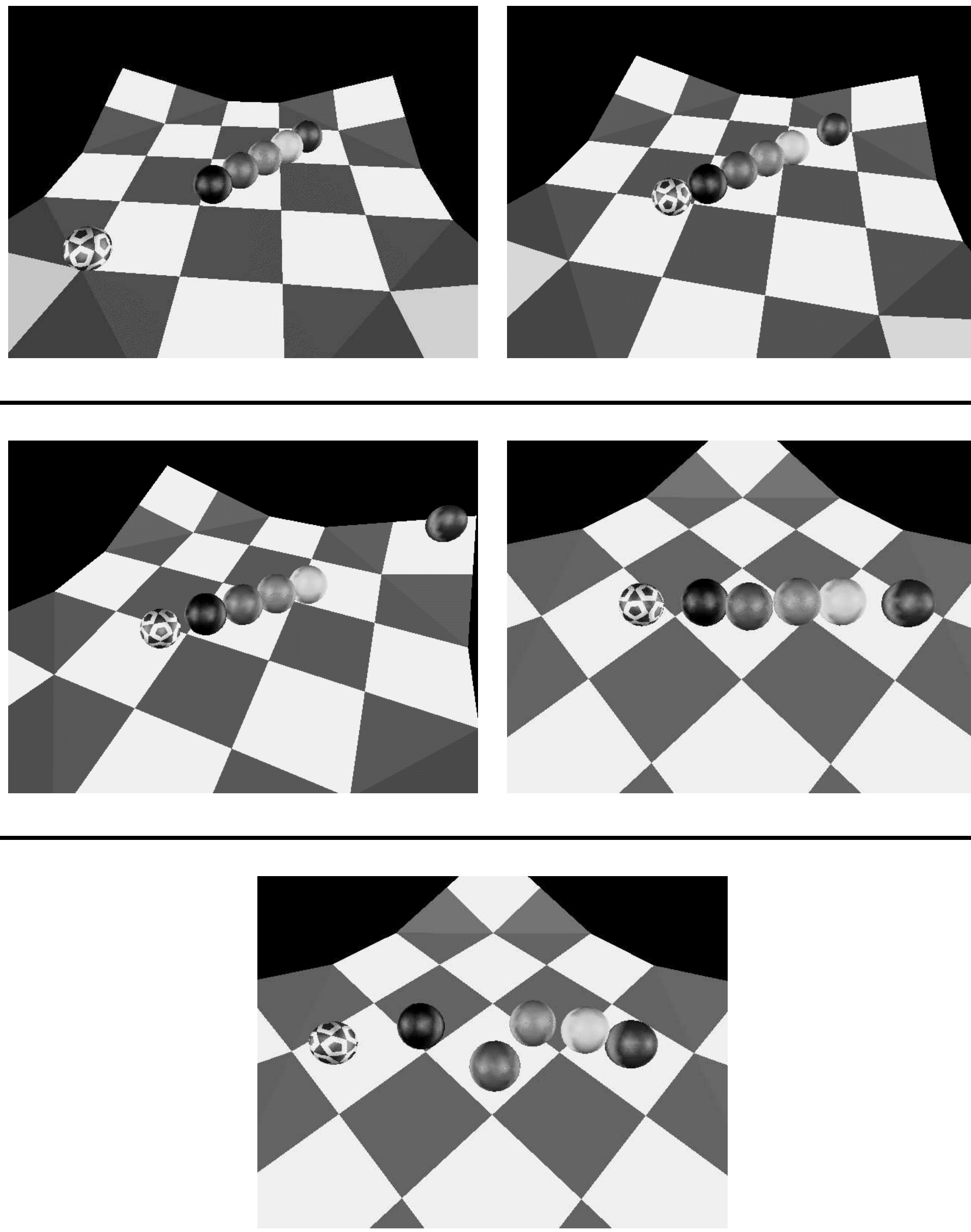

Figure 7.4: *Snapshots from the* chain of balls *simulation.*

Chain of balls

The *chain of balls* simulation illustrates the propagation of an impulse through a chain of contacting bodies (Figure 7.4). Four balls are initially placed a collision epsilon apart, so that if any ball moves toward another, it will immediately penetrate the collision envelope. After the striking ball hits the chain, the ball at the other end of the chain rolls away, while the rest remain basically motionless. The moving ball rolls up a small ramp, then back down to strike the chain. The second collision impulse is propagated less cleanly because the balls have drifted slightly from their straight line configuration. The drift is primarily a result of the polyhedral models used for the spheres, which prevent the collisions from being exactly central.[1]

Under analytic approaches, there are two ways to propagate impulses through chains of contacting bodies. One method computes the impulses one at a time, and the other computes them simultaneously. The latter is computationally preferable in some situations, but can generate non-physical results in others. For instance, when the simultaneous model is used for a ball chain simulation, the striking ball launches the entire chain into motion, while it recoils backwards after impact [CS89, Bar89]. How to choose between these models is not clear. There is no ambiguity in the impulse-based model, which produces the physically correct result.

Balls in dish

The *balls in dish* simulation (Figure 7.5) involves dropping seven balls into a shallow dish to let them interact at will. After bouncing around for a few moments, the balls come to rest with one ball at the center of the dish, surrounded symmetrically by the other six. Although *Impulse* does not explicitly try to minimize the energy of the system, this is the macroscopic result as collisions dissipate energy from the system.

In the final configuration, there are 12 simultaneous contacts within the cluster of balls, plus seven contacts between the balls and dish, which are all easily maintained with microcollisions. These contacts are easier to model via collisions than those for the *block drop* simulation (described above) for two reasons. First, there are not stacks of objects involved; because of the shallow dish angle, the balls are not pushed into one another as strongly.

[1] A central collision is one in which the collision point is on the line segment joining the centers of mass of the colliding bodies.

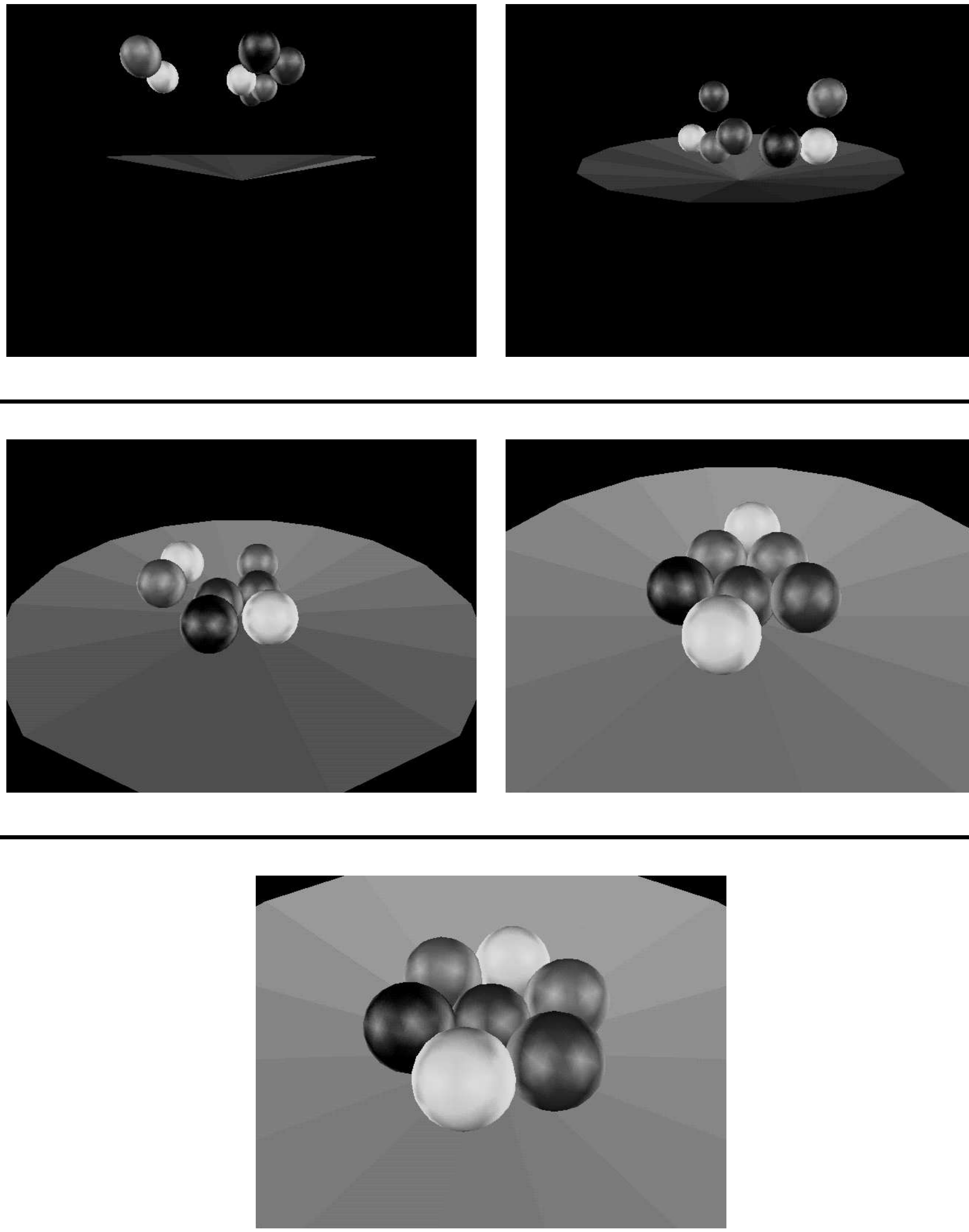

Figure 7.5: *Snapshots from the* balls in dish *simulation.*

Second, collisions between spheres are central, and impart little angular acceleration to the colliding bodies. As a result, the TOI bounds are much tighter. In contrast, when one corner of a cube collides with a large flat surface, the angular acceleration of the cubes cause the opposite corner to strike the surface soon afterward.

Coins

The *coins* simulation (Figure 7.6) illustrates the wide variety of macroscopic contact modes that are produced by *Impulse's* collision model. In this simulation, four coins are rolled toward the center of a plate, and four coins are simultaneously tossed into the center of the plate. All of the coins collide in the middle. Through the course of the simulation, many types of contact occur, including: colliding, sliding, prolonged rolling, resting contact between different coins, and resting contact between the coins and plate. Also demonstrated is the familiar motion of a coin rolling along a path of increasing curvature, eventually falling over and "rattling" to rest. The coin system is very representative of the types of physical systems which are well suited to impulse-based simulation.

Pool break

Figure 7.7 is a series of snapshots from the simulation of a pool break. Like the *chain of balls* simulation, this simulation entails propagating collision impulses through chains of contact, but with a more complex initial contact configuration: there are initially 30 contact points between the rack of 15 balls. The balls in the pool break simulation are initially placed one collision epsilon apart, so that any motion on the part of one ball toward another results in an immediate collision.

A high number of collision checks must be processed at the moment of the initial break. Figure 7.8 shows the number of collision checks per second of simulation time. After the balls begin to separate, the collision check frequency quickly drops as higher TOI bounds are achieved. The collision check rate stabilizes at approximately 3 kHz after the break is over. These collision checks are between the balls and the surface of the pool table, and are needed to keep the balls from penetrating the table.

The collision model supports spin on the balls. By changing the initial angular velocity of the cue ball, it can be made to draw back or eventually roll forward after the initial impact with the rack. Other balls follow paths which deviate slightly from a straight

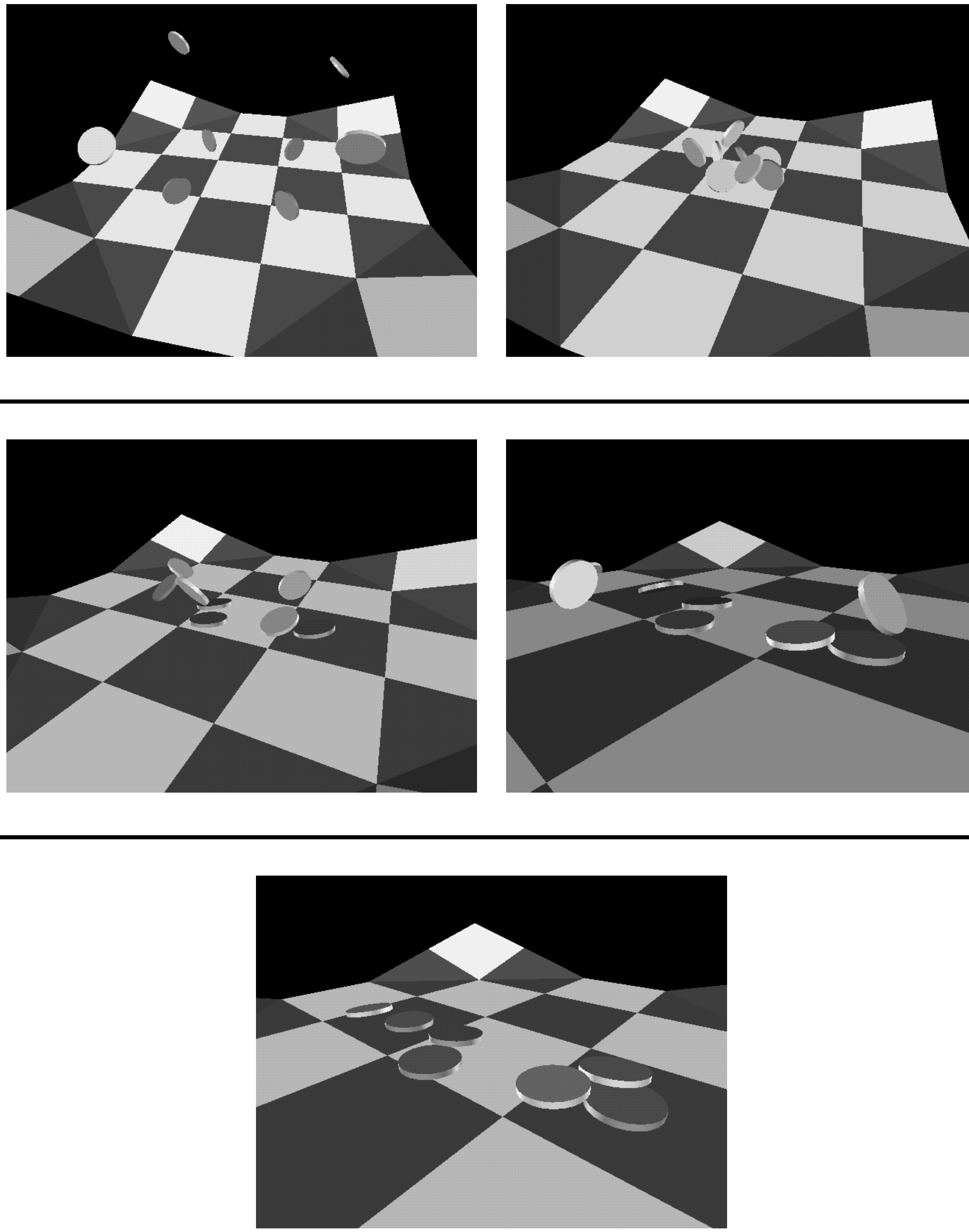

Figure 7.6: *Snapshots from the* coins *simulation.*

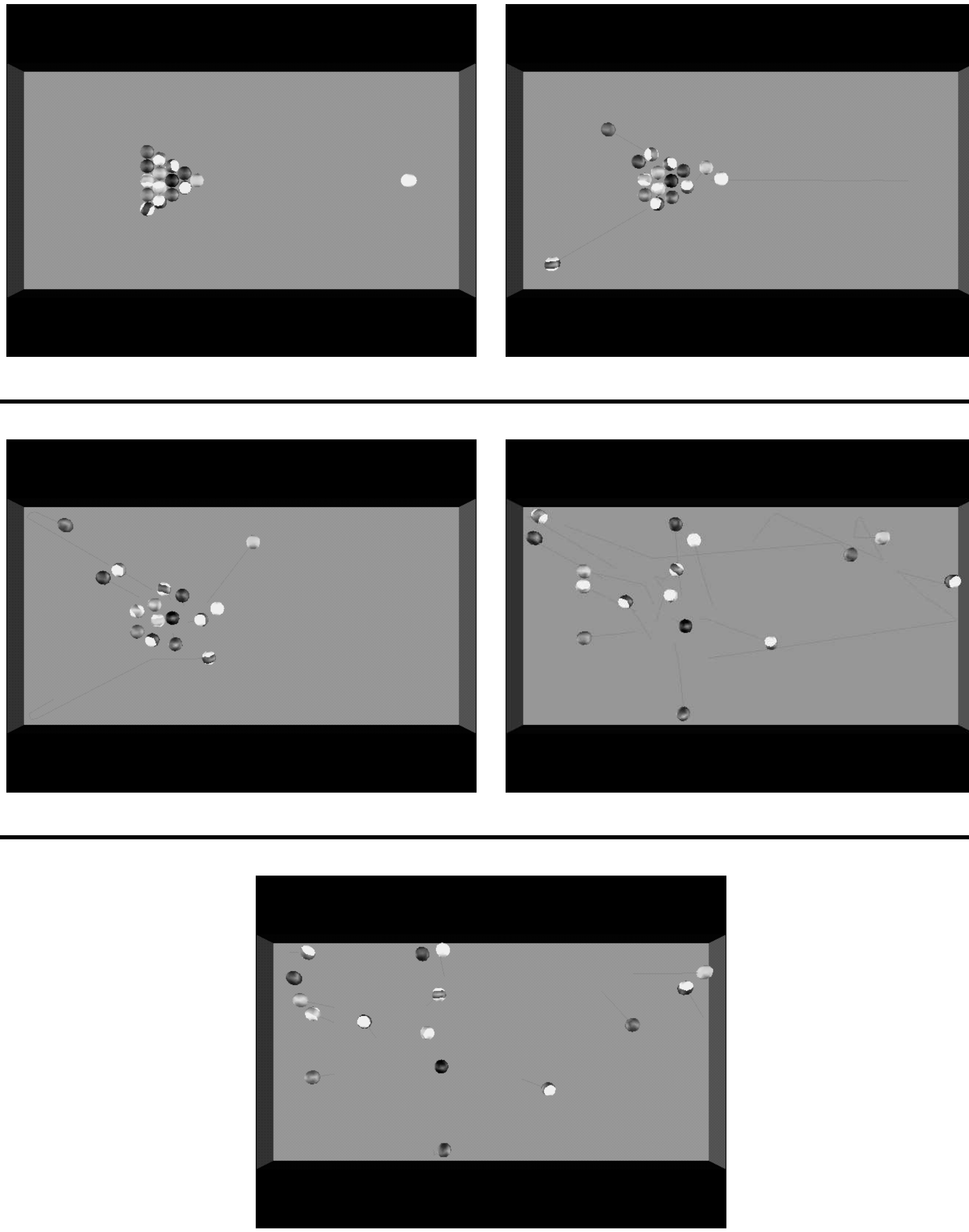

Figure 7.7: *Snapshots from the* pool break *simulation.*

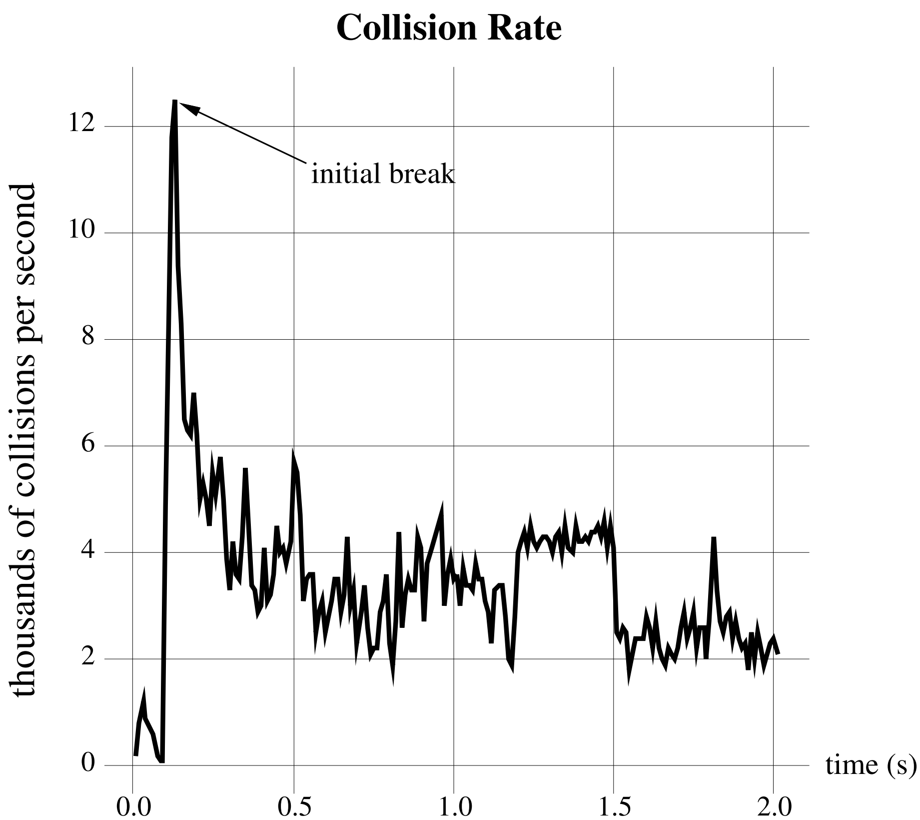

Figure 7.8: *The frequency of collision checks as a function of time during a pool break.* Time *refers to the time in the simulation, not the execution time required to compute the simulation.*

line, due to spin.

Bowling

A typical ten-pin bowling throw involves both prolonged and transient contact modes. The ball is lofted onto the alley, and settles into continuous contact with the alley, sliding at first. Eventually, contact with the alley causes the contact mode to shift from sliding to rolling, as the relative tangential velocity at the contact between the ball and alley is brought to zero. If the bowler imparts the proper initial translational and angular velocities to the ball, the sliding phase will bring the ball toward the gutter, and the rolling phase will cause it to hook back toward the center of the alley. Once the ball strikes the pins, the contacts are predominantly transient, with many collision occurring. Snapshots from such a bowling simulation are shown in Figure 7.9.

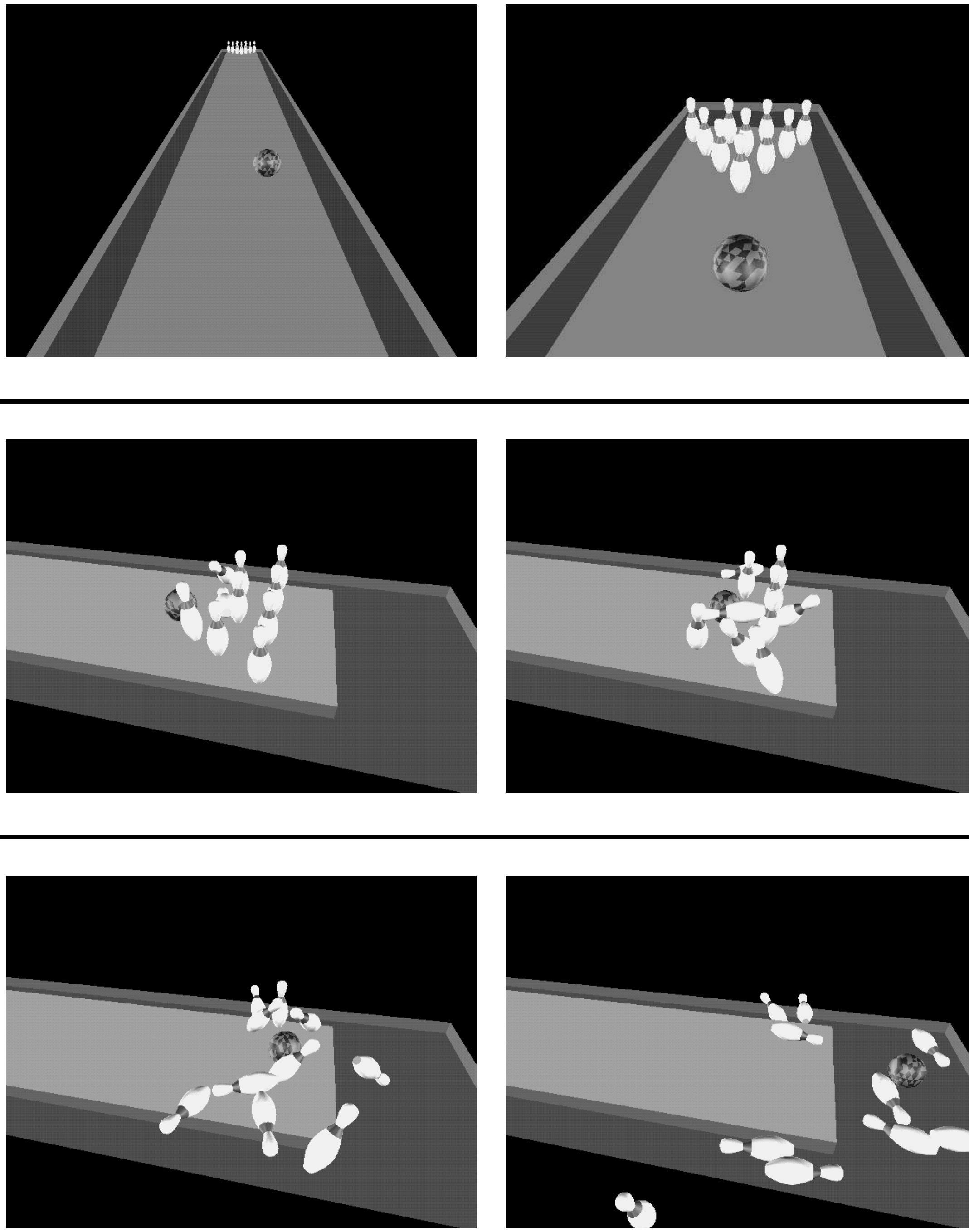

Figure 7.9: *Snapshots from the* bowling *simulation.*

A series of bowling experiments was performed to test *Impulse's* ability to obtain physically valid results. In the first batch of simulations, a straight ball was thrown down the alley by launching the ball with zero angular velocity, and a center of mass velocity in the $+y$ direction (Figure 7.10). 320 trials were performed, keeping the initial ball velocity

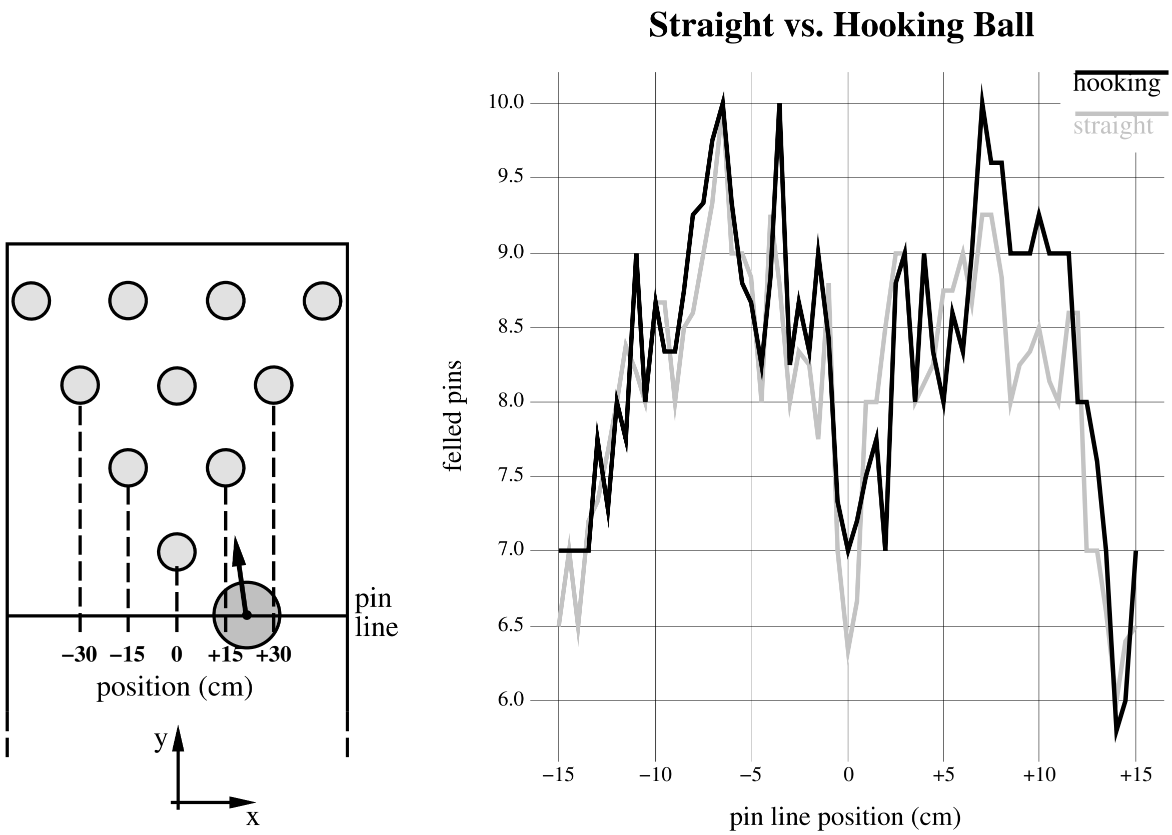

Figure 7.10: Left: *Bowling experiments were performed by recording the horizontal position of the ball as it crossed the pin line, and the number of pins that were knocked down.* Right: *Results of the bowling experiments for straight and hooking balls.*

constant, but varying the initial x-coordinate of the ball's center of mass over a 40 centimeter window. In a second batch of 320 trials, the initial ball velocity conditions were altered to produce a right-hander's hooking ball: angular velocity of -12 rad/s in the $+y$ direction and a linear velocity at an angle of $-2°$ from the y-axis. Accurate physical dimensions and masses were used for all bodies, except for a slight approximation in the shape of the pins.

Figure 7.10 shows the number of felled pins versus the ball position as it crossed the pin line (ordinates are averaged over 5 mm wide abscissa windows). The hooking ball is slightly better than the straight ball at most positions along the pin line, and is significantly better over a range between the head pin and rightmost second row pin (+6 to +12 cm on the pin line). This agrees with the accepted wisdom that a right-handed bowler's best strategy is to throw a hooking ball between these two pins. The plots also illustrate the dip in felled pins due to splits, when the ball hits the head pin dead on. Obtaining these results analytically would be nearly impossible, but the chaotic bowling system is an ideal application for impulse-based simulation: the evolution is collision intensive, with many transient contacts between objects, and there is a gradual shift in contact mode between the ball and the alley (bouncing to sliding to rolling).

Ball on spinning platter

Consider a ball rolling on a spinning platter. The nonholonomic model and the Vakonomic model are two variational-based approaches to deriving the dynamics of this system, which yield different answers. Lewis and Murray computed simulations based on each of these models, and then performed experiments to test their accuracy, using a large steel ball on a spinning plexiglas surface [LM95]. They note that the general experimental behavior was in good agreement with certain nonholonomic simulations, such as the one pictured in Figure 7.11. Also in the figure is a snapshot from a similar experiment using *Impulse*. Here, the nonholonomic contact between the ball and platter is modeled with the usual collisions. The trajectories generated this way qualitatively match those of Lewis and Murray. This example illustrates the versatility of the impulse-based method: with no extra effort, collisions model the nonholonomic constraint between the ball and platter in a physically accurate way.

Rattleback top

The *rattleback top* simulation demonstrates the ability of the impulse-based method to model nontrivial contact interaction. A rattleback top is an elongated solid with a convex bottom, much like the hull of a ship. If spun in its preferred direction, the top will continue to spin in that same direction until friction eventually stops it. If spun in the other direction, however, instability soon occurs as the top begins to rock back and forth, and eventually reverses its spinning direction. This anomalous behavior results from a slight asymmetry in the top, either in the shape of the bottom or in the mass distribution. The key requirement is that the principle axes of curvature at the contact point must be skewed in a particular

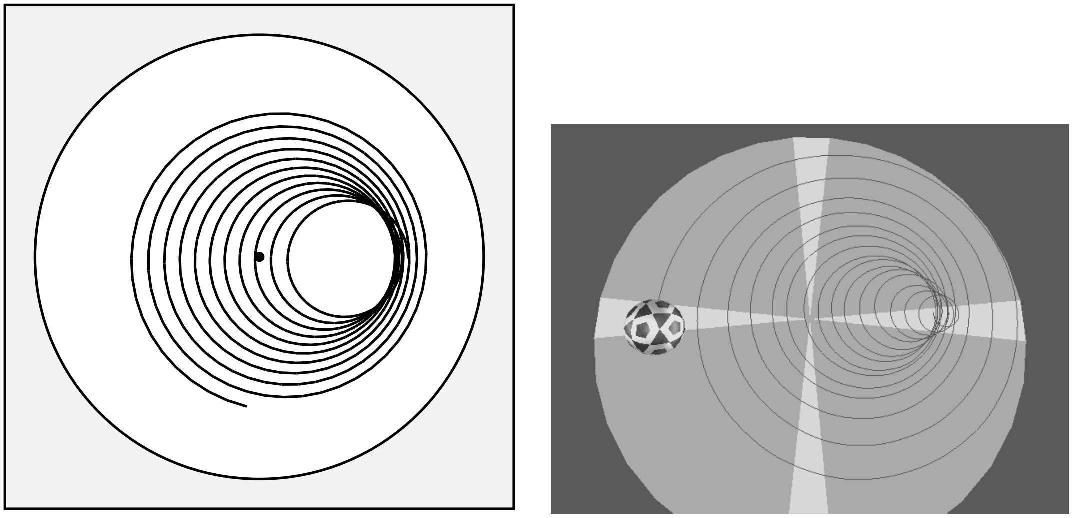

Figure 7.11: Left: *Trajectories of a ball on a spinning platter simulated by Lewis and Murray; the physical experiments generally exhibited this behavior [LM95] (reprinted with permission).* Right: *A snapshot of an analogous simulation using* Impulse*; the spiral lines indicate the ball's trajectory.*

manner relative to the principle axes of inertia of the top.

A rattleback experiment was performed to test *Impulse's* ability to predict this phenomenon. First, two identical, balanced, ellipsoids were spun with the same initial conditions to verify that they remained in phase when spun. The mass of each ellipsoid was 500 grams. Next, a 30 gram point-mass was attached to the surface of one of the ellipsoids, in a position which skewed the principle axes of inertia in the prescribed way. The experiment was redone, spinning the ellipsoids in a manner such that the rattleback was spun in its non-preferred direction. After approximately two revolutions, the rattleback top began rocking, and quickly reversed its spinning direction, as predicted. Snapshots from the simulation are shown in Figure 7.12.

Part feeder chute

The *parter feeder chute* simulation is one of a number of simulations related to part feeding. Six parts are dropped into an inclined tray, funneled through a narrow opening in the lower end, and land on the flat surface below. The parts are modeled after an actual industrial part: a non-convex insulator cap. The parts slide along each other in the chute, before spreading out as they fall onto the surface (Figure 7.13). Experiments like

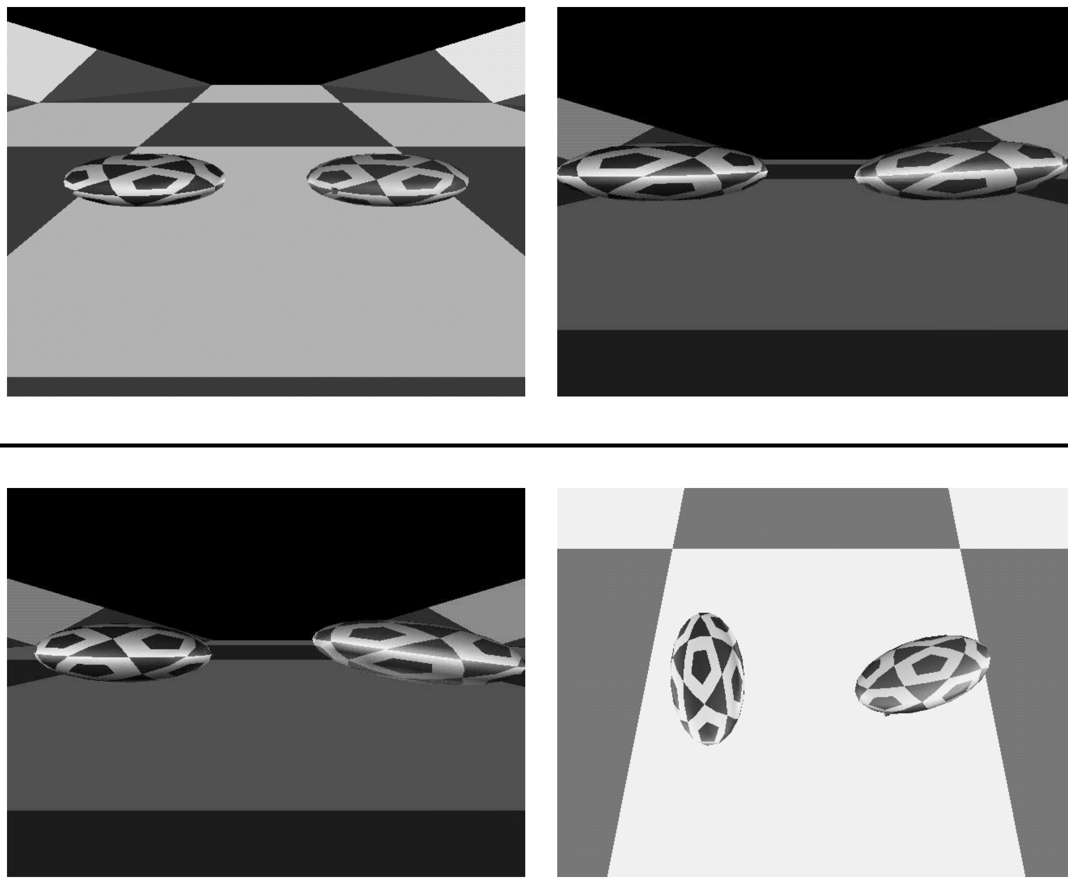

Figure 7.12: *Snapshots from the* rattleback top *simulation.*

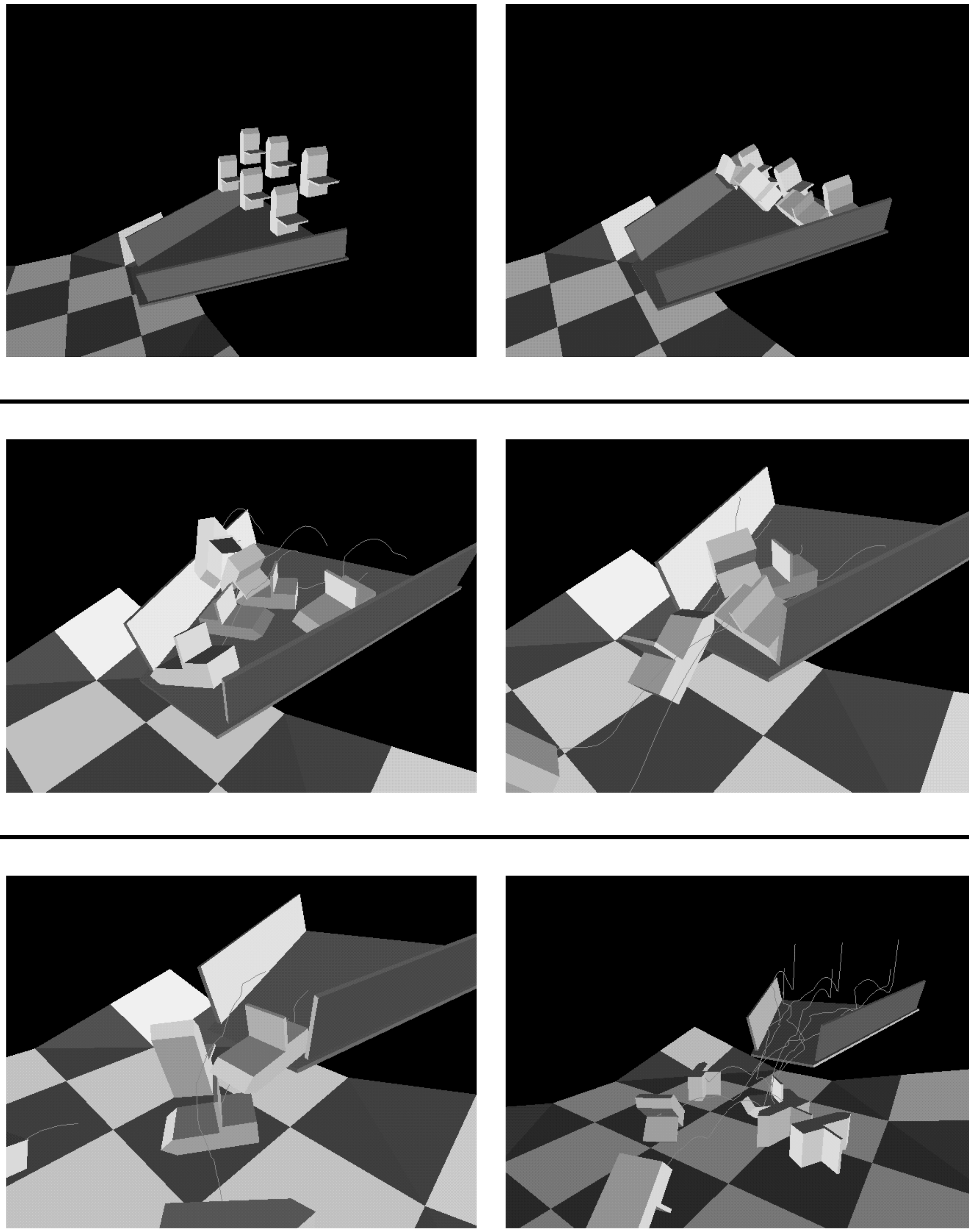

Figure 7.13: *Snapshots from the* part feeder chute *simulation.*

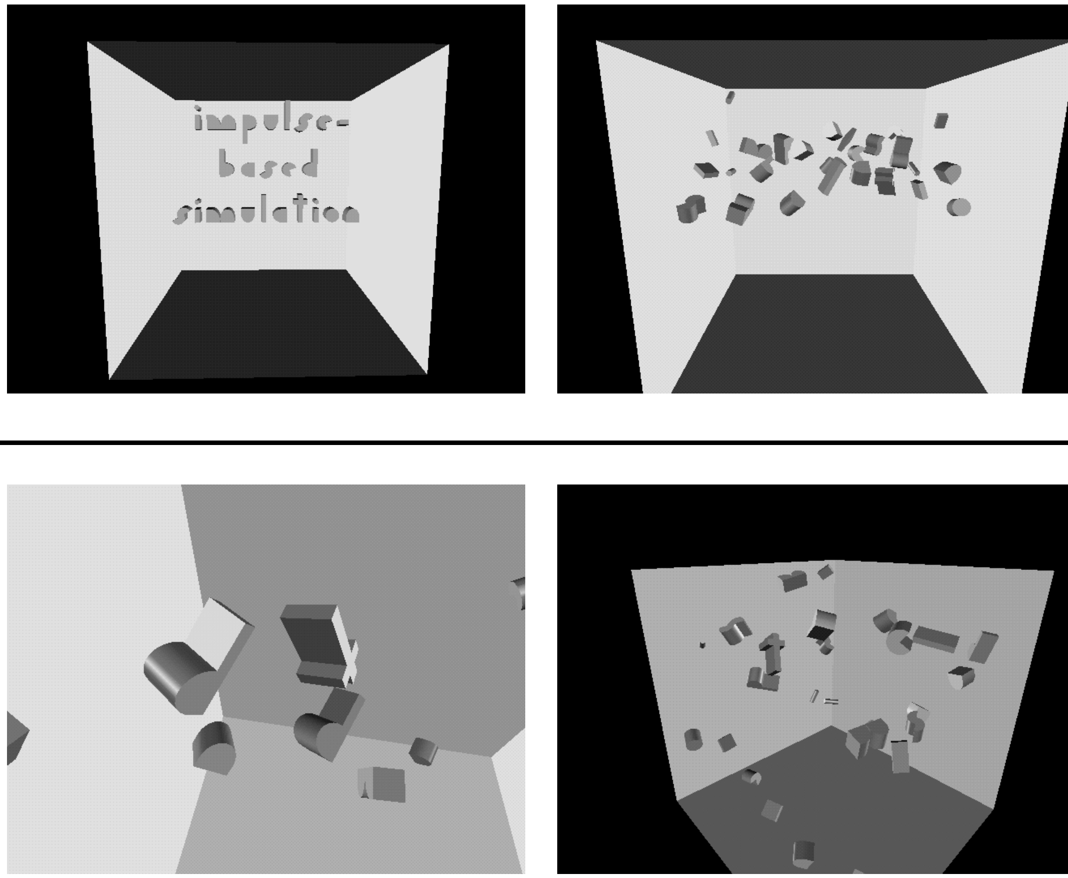

Figure 7.14: *Snapshots from the* exploding text *simulation.*

this can generate stochastic data related to a particular feeding operation. The data can determine the feeder's susceptibility to jamming, and its ability to singulate the parts. Such an application, using *Impulse*, is described in [BC96].

Exploding text

Physically based modeling is not only a valuable tool for simulation and virtual prototyping, but also for generating animations. One example is the *exploding text* simulation. The letters of the words "impulse-based simulation" are given random initial velocities, and set free to move through the space in an enclosed volume, with gravity turned off. (Figure 7.14). This example is a good test of the collision detection system: there are 26 independently moving rigid bodies, many of them non-convex. It illustrates the use of simulation to produce interesting visual effects. The animation is even more compelling when played back in reverse: from random debris the words "impulse-based simulation" materialize.

7.2 Passive hybrid simulation

As discussed in Chapter 5, pure impulse-based methods are not suitable for modeling tightly constrained contact, such as that occurring at joints. Hybrid simulation allows for certain contacts to be modeled as joints, greatly expanding the range of systems that can be simulated. This section describes three examples that use passive hybrid simulation. *Passive* means that there are no control systems acting. All external forces on multibodies are due to gravity, collisions, or passive elements like springs and dampers.

See-saw

In the *see-saw* simulation, eight balls are placed in a shallow, funnel-like tray, and roll toward a rectangular opening in the center of the tray. Blocking this opening is a see-saw: a one link multibody, fixed to the tray through a revolute joint. The balls eventually land on the see-saw, causing it to turn and rock as they pass through the opening (Figure 7.15). Contacts between different balls, between the balls and the tray, and between the balls and see-saw are all modeled with collisions.

Part sorter

The *part sorter* simulation is another application from the part feeding domain. Six parts, of three different types, slide down tracks arranged in a switchback fashion. Two trap doors in the tracks allow parts with sufficient mass to fall through to the track below; lighter parts slide across the trap doors. In this way, parts are sorted based on their mass (Figure 7.16). The trap doors are single link multibodies connected to the fixed tracks through hinges. The hinges are equipped with springs to hold the doors closed. The spring constants were chosen to let certain parts fall through, but let others pass over. Dampers

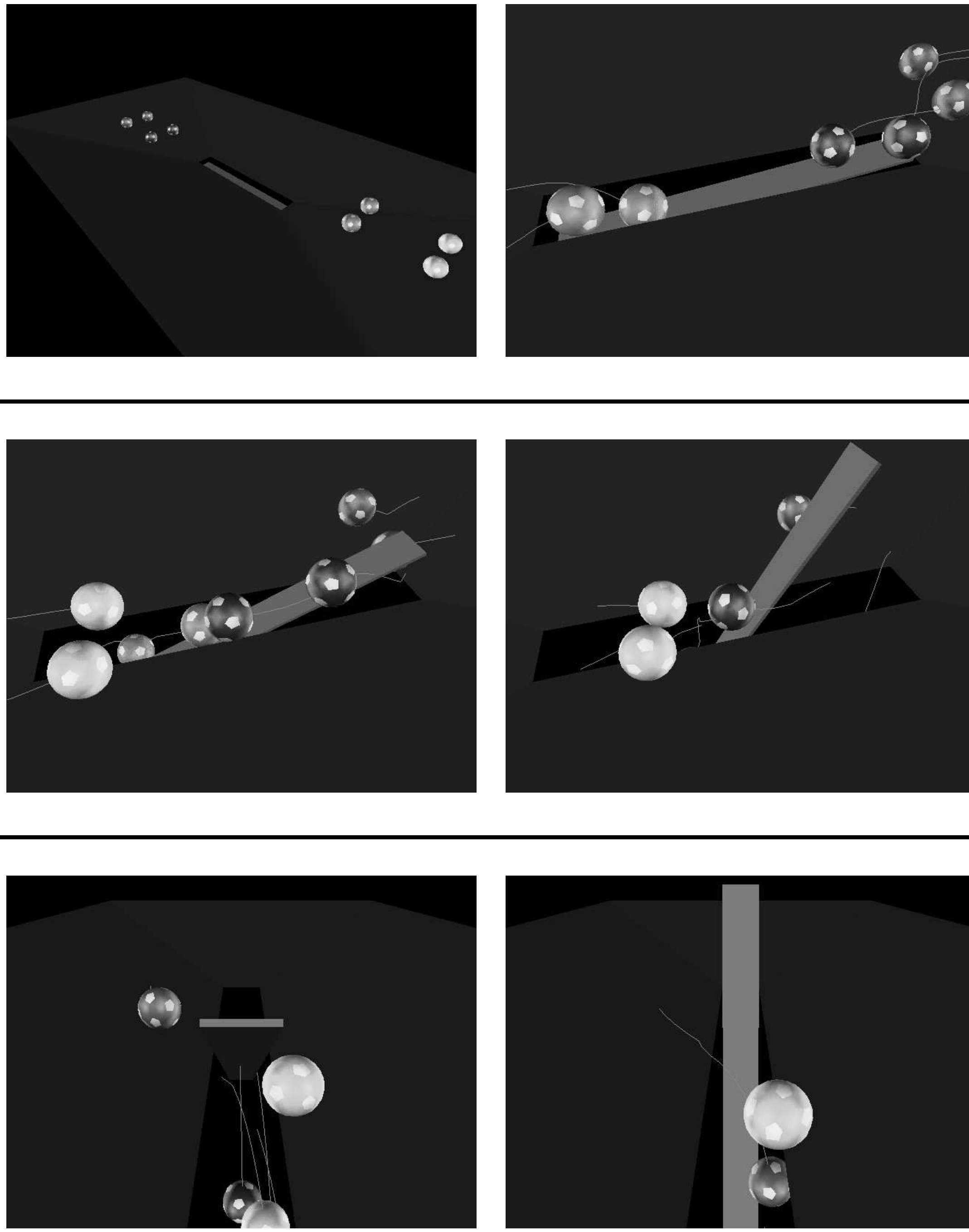

Figure 7.15: *Snapshots from the* see-saw *simulation.*

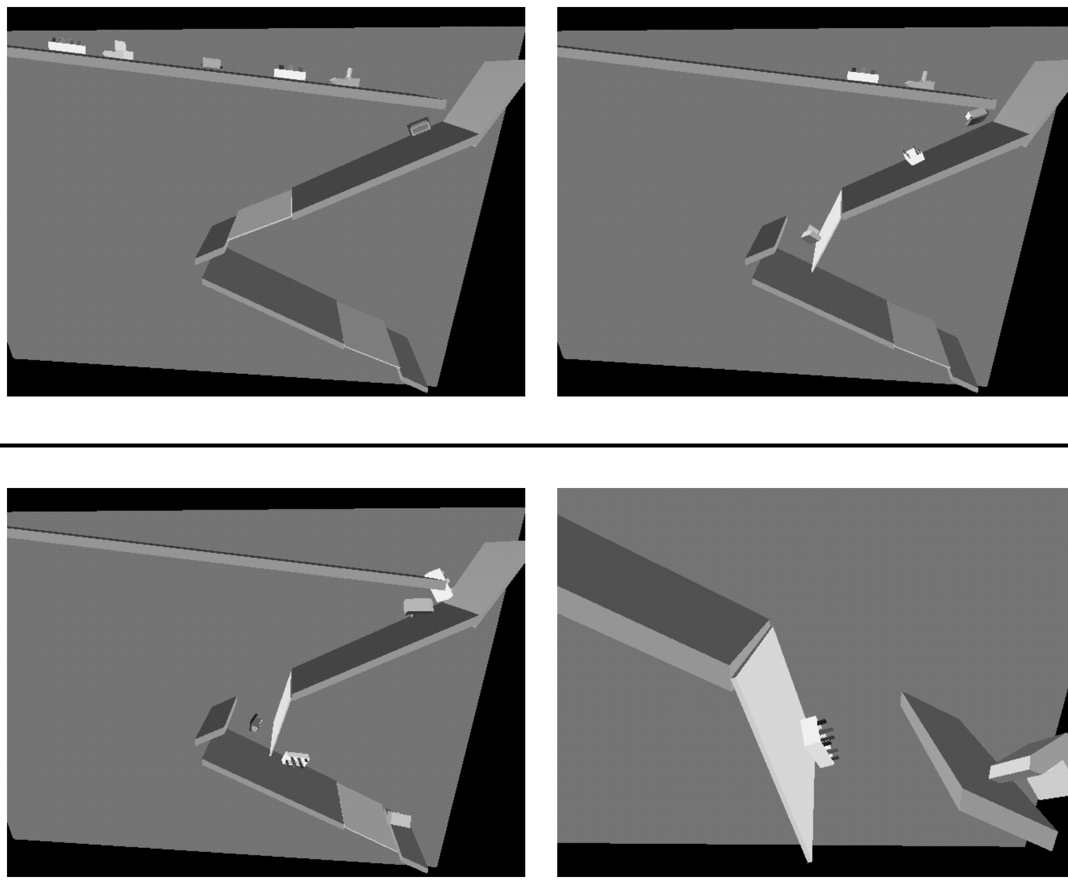

Figure 7.16: *Snapshots from the* part sorter *simulation.*

are attached at the joint to prevent excessive oscillation of the doors. Interaction between the parts and the tracks and trap doors are modeled with collisions; the hinge contacts are modeled with constraints.

The spring constants are adjusted so that *individually* one part type glides across the first trap door, one part type falls through the first trap door but glides across the second, and one part type falls through both trap doors. In practice, the simple sorter performs poorly because its dimensions are too small. Parts can not be adequately separated, so the trap doors do not have sufficient time to return to their resting state between parts. This type of information would be very useful to the designer of a part sorter.

Triple pendulum

A final example of passive hybrid simulation is the *triple pendulum* simulation (Figure 7.17). Six balls are dropped into a shallow dish. Suspended above the dish is a pendulum with three links and three revolute joints. The last link ends with a mallet. This pendulum is positioned so that it can never hit the dish, but so that it can reach the balls in the dish. It is not actuated, but dropped from a high potential energy state, so that gravity drives it. As it swings, it collides with the balls, occasionally knocking one out of the dish. The interactions between pairs of balls, between the balls and the dish, and between the balls and the pendulum are modeled with collisions. Eventually, the pendulum comes to rest in its minimum energy state, with the mallet resting between the three remaining balls. The pendulum is not damped; its decrease in energy results entirely from collisions with the balls.

7.3 Controlled hybrid simulation

Hybrid simulation is not restricted to passive systems. External controllers may also be attached to objects, making the simulator a powerful tool for animation, as well as simulation and testing. Three examples using *Impulse* are discussed below.

Furniture arrangement

The *furniture arrangement* simulation can be viewed as an animation application, or as a design application with intelligent objects. This example only used rigid bodies—a

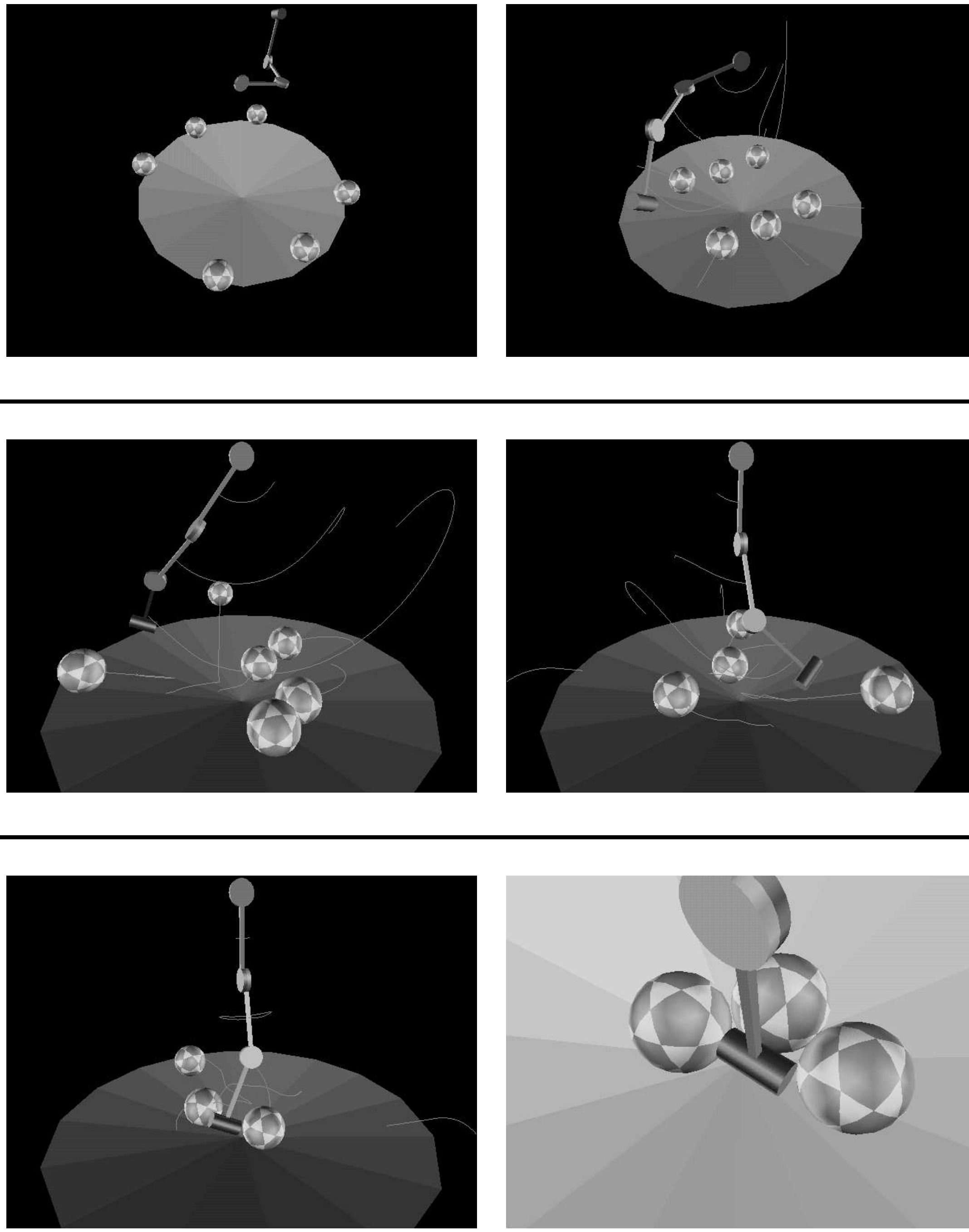

Figure 7.17: *Snapshots from the* triple pendulum *simulation.*

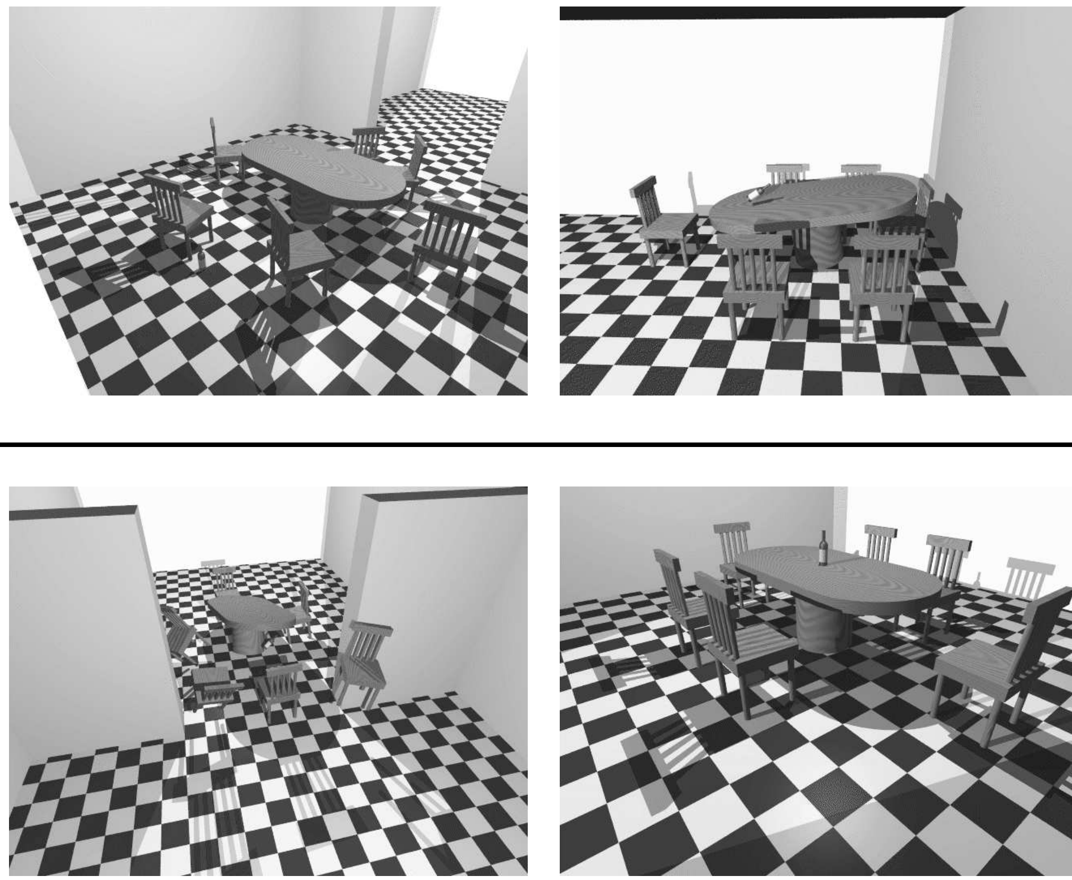

Figure 7.18: *Snapshots from the* furniture arrangement *simulation.*

trivial case of multibodies—and does not illustrate the full benefits of hybrid simulation. But it is a good test of the control architecture. The moving objects are a table, six chairs, and a wine bottle that move through two adjoining rooms (Figure 7.18). The objects are all controlled by applying external forces and torques via the control architecture. The table controller steps through a sequence of eight user specified configurations, applying forces and torques to gravitate the table toward each configuration. Damping forces and torques are also applied. When a configuration goal is attained, and the table velocity subsides, the control system advances to the next goal.

Each chair is driven to gravitate toward a goal configuration specified relative to the table, again using a PD controller; as the table moves, the chairs attempt to follow it. Chair goal positions are also modified by the presence of nearby walls, so that they "scoot in" if the table moves close to a wall. Finally, the bottle has a simple control law which repeatedly causes it to jump onto the table, if not there already, and gravitate toward the center of the table. To affect a jump, the controller computes initial velocity values which would arc the bottle in a parabolic trajectory onto the table, and writes these new velocity values to the bottle's state, as if it had experienced a collision. This illustrates the flexibility available to the control designer. All object controllers also apply torques to keep the objects upright.

A total of four controllers are required: one for the table, one for the bottle and two for the chairs (one to control them to goal positions, and one to update these positions in the presence of walls). Each controller is a single C function, about one page long. The controller rates vary from 20–100 Hz.

Although real furniture is no more autonomous than a Luxo-lamp, this example demonstrates the ease with which interesting motion can be produced from simple control systems built on top of a dynamic simulator. The entertaining animation can be viewed as the desired product of the simulation. From a more serious perspective, this example illustrates the use of simulation as a design or layout tool. Consider the labor intensive job of arranging furniture in a large, architectural virtual environment (as in [BS95]). The task is made easier if the user only specifies the positions of certain objects, like tables, and other objects, like chairs, adjust themselves accordingly. The chairs are *intelligent objects*, with useful behaviors.

Bicycle

The *bicycle simulation* involves an unmanned bicycle guided over an obstacle course (Figure 7.19). A controller designed to balance the bike at a specified roll angle was provided by Getz [Get94]. In contrast to the bike controller employed by Hodgins, *et. al.* [HWBO95], Getz's control law is nonlinear, employs a dynamic model of the bicycle, accounts for the nonholonomic constraint between the wheels and ground, and does not decouple the control of the pedals and handlebars. Hodgins, *et. al.* modeled a rider on the bike, and built basic obstacle avoidance and clustering abilities into the control law; these

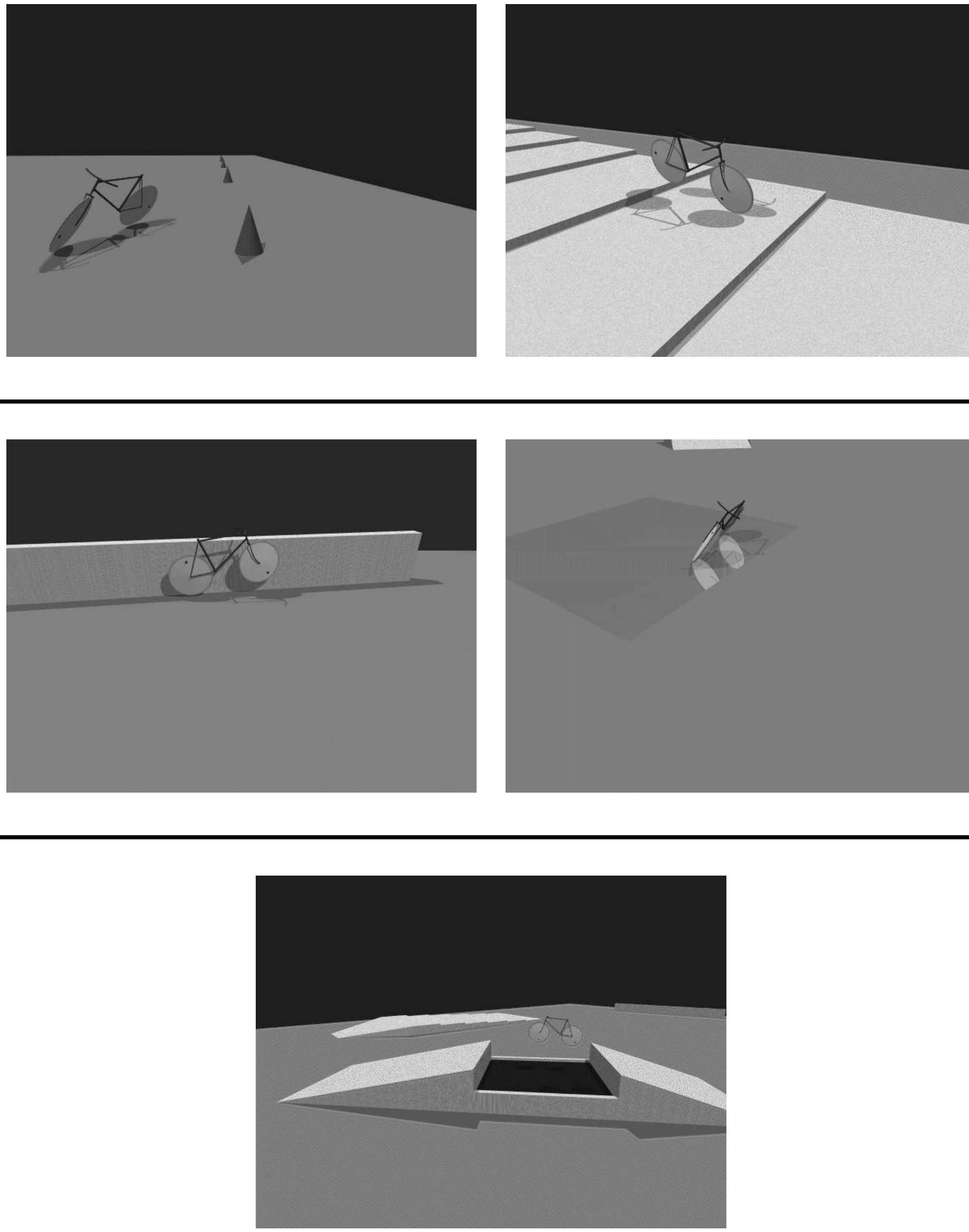

Figure 7.19: *Snapshots from the* bicycle *simulation.*

are not present in Getz's controller.

This example was chosen for two reasons. First, Getz's control law relies on correct modeling of both the bicycle dynamics and constraint forces with the road in order to function properly. Second, the controller was developed independently, without regard to any particular style of dynamic simulation. Prior to this experiment, the control law had been implemented only as a MATLAB simulation of a bicycle on a flat surface, using idealized motion constraints, and without collisions or obstacles.

The bicycle is a 4-link multibody: a frame, a fork, and two wheels; geometry and masses are based on a standard 26" bike. The kinematic topology can be described as a serial linkage, but the model was built as a tree linkage with the frame as the floating base link (Figure 7.20). The controller takes as input the current steering angle, roll angle, rear

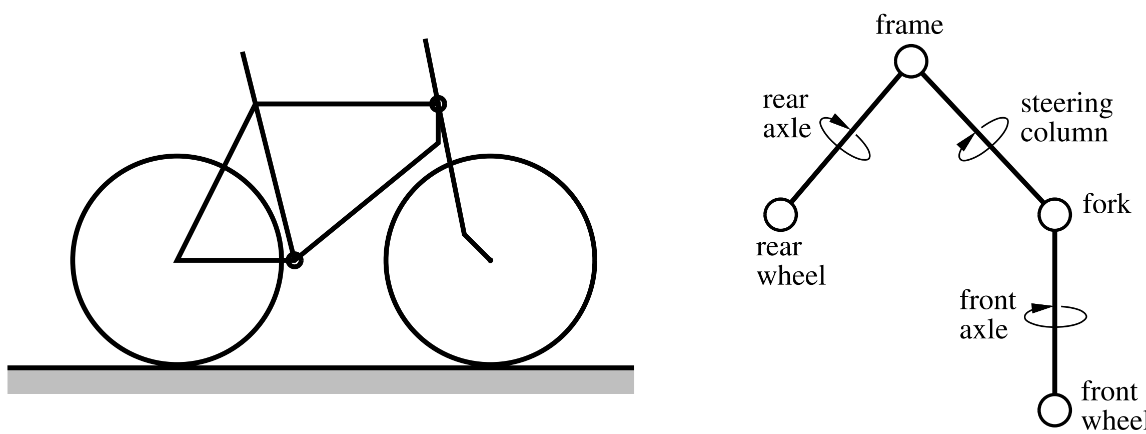

Figure 7.20: *The kinematic structure of the bicycle.*

wheel velocity, and the derivatives of these quantities. Using a computed torque method, it calculates desired accelerations of the pedals and handlebars to achieve the user defined roll angle and forward velocity profiles. The front wheel is not actuated and rolls freely. Getz's MATLAB code was translated in essentially unchanged form to a C function called by *Impulse's* control architecture, at a 100 Hz rate. With this control law and the defined profiles, the bike is guided over an obstacle course which includes a pylon slalom, stair steps, a wall, an ice patch, and a jump.

This example demonstrates how control strategies may be tested under more varied conditions with an impulse-based simulator that supports general collision and contact modeling. Hazards or other agents can be placed into the simulation without any modification of the control law; a good controller causes the system to react in a natural way. The graphs shown in Figures 7.21 and 7.22 show how well the controller worked in the presence of the various disturbances.

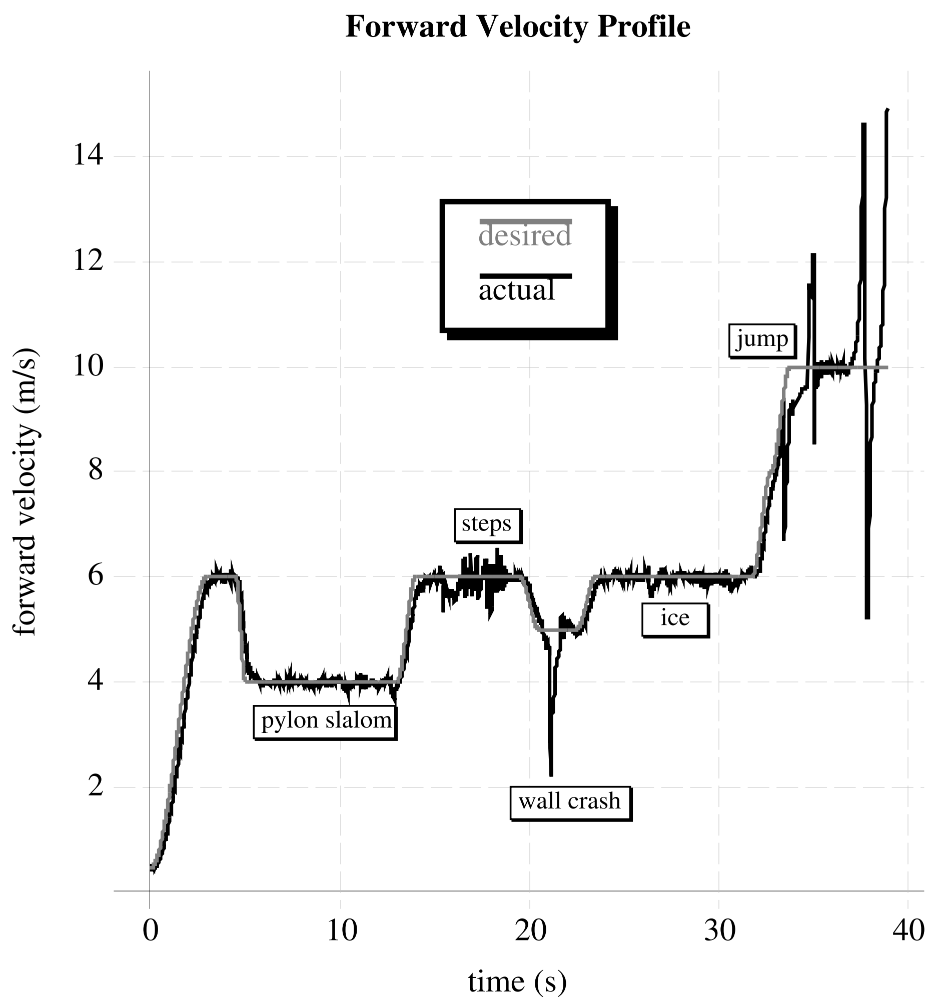

Figure 7.21: *The forward velocity profile as the bike traversed the obstacle course.*

Creature war

The *creature war* simulation provides a detailed test of the multibody collision response algorithms, and the control support architecture. In this test, a rover vehicle chases crawling bugs, annihilating them by pushing them off the edge of the world (Figure 7.23). Each bug has an ellipsoidal body and two elliptical legs (Figure 7.24). The leg velocities

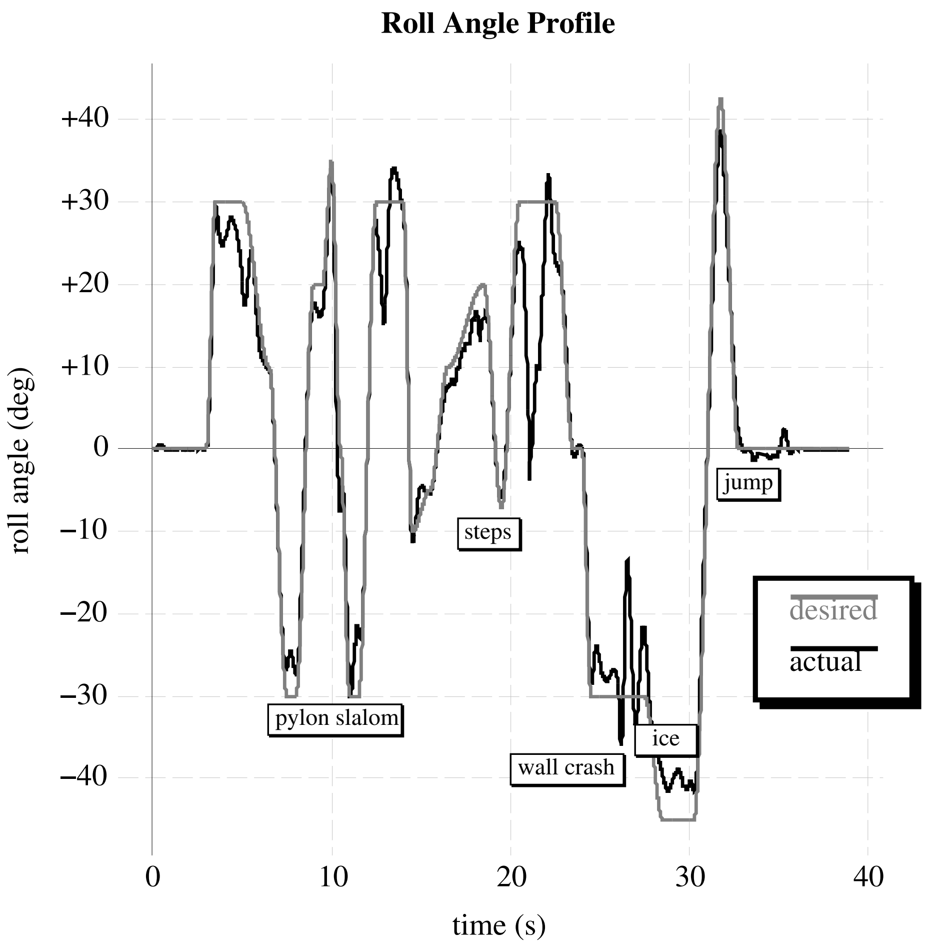

Figure 7.22: *The roll angle profile as the bike traversed the obstacle course.*

are regulated with individual PD controllers. Bugs control their headings by commanding different velocities to their legs. Bug behaviors are generated by a high level controller that computes a desired heading using a potential field approach. They have a preference to be in the center of the square world, and a stronger preference to avoid the rover when it is nearby. Each bug's high level controller uses the bug's current position and that of the rover to compute a new desired heading.

The rover is a 7-link multibody (Figure 7.25). The front wheels turn on axes connected directly to the chassis, while two prismatic joints allow each rear wheel axis to move vertically, relative to the chassis. These prismatic joints are equipped with shock absorbers (springs and dampers) to smooth out bumps in the terrain. The rover uses four PD controllers to regulate the velocity of its wheels. It changes its heading by commanding

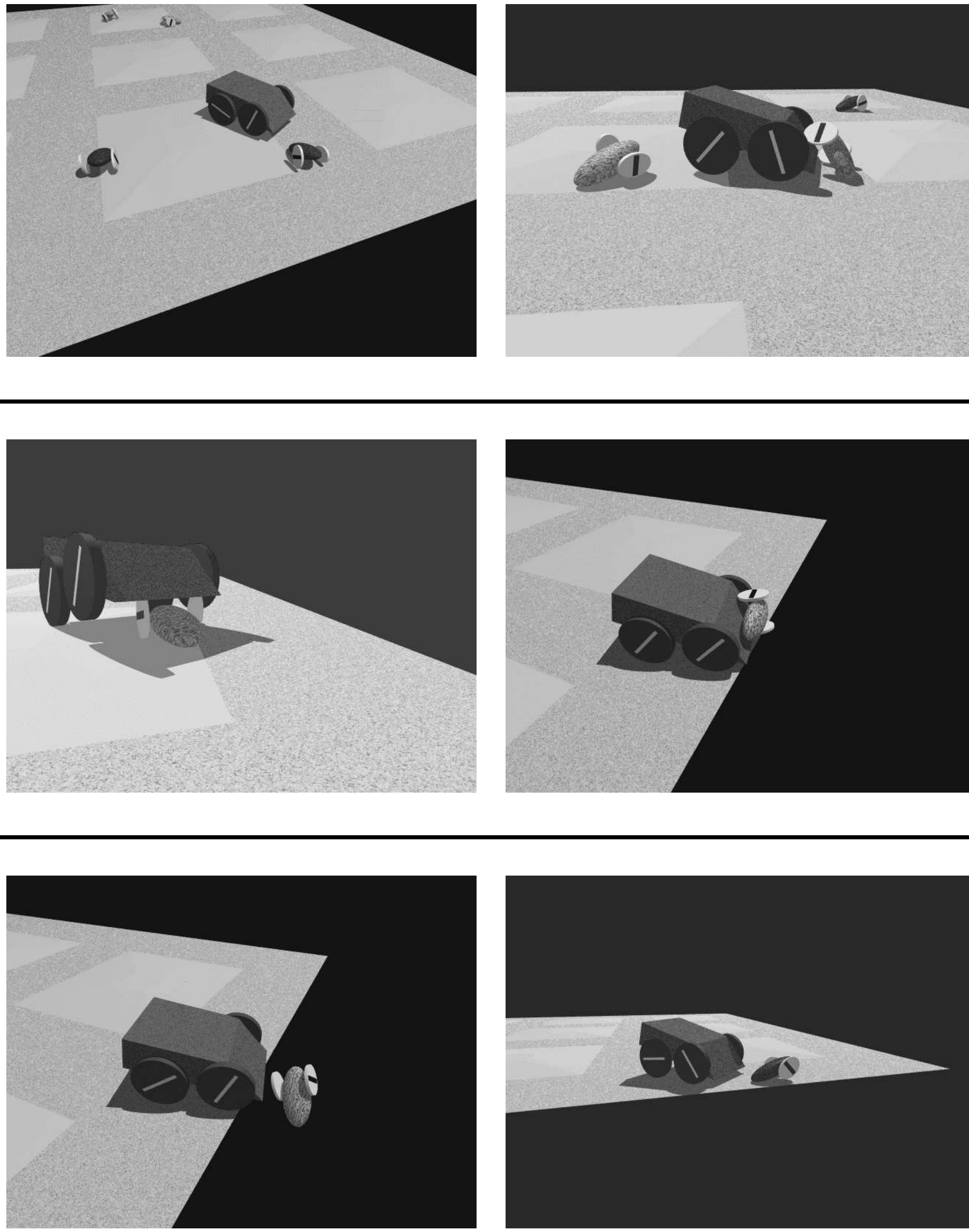

Figure 7.23: *Snapshots from the* creature war *simulation.*

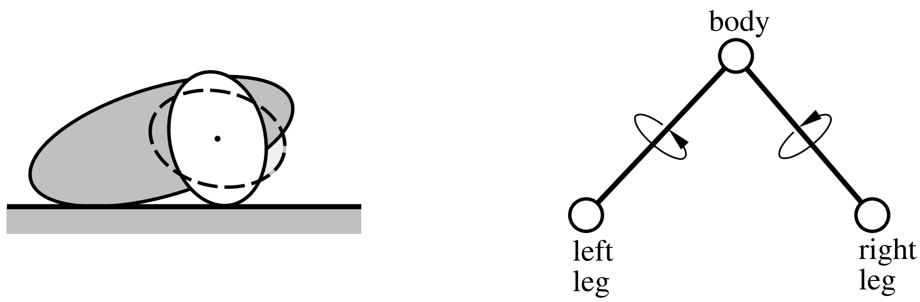

Figure 7.24: Left: *Side view of a bug.* Right: *Kinematic tree for a bug, showing the individual links and joints.*

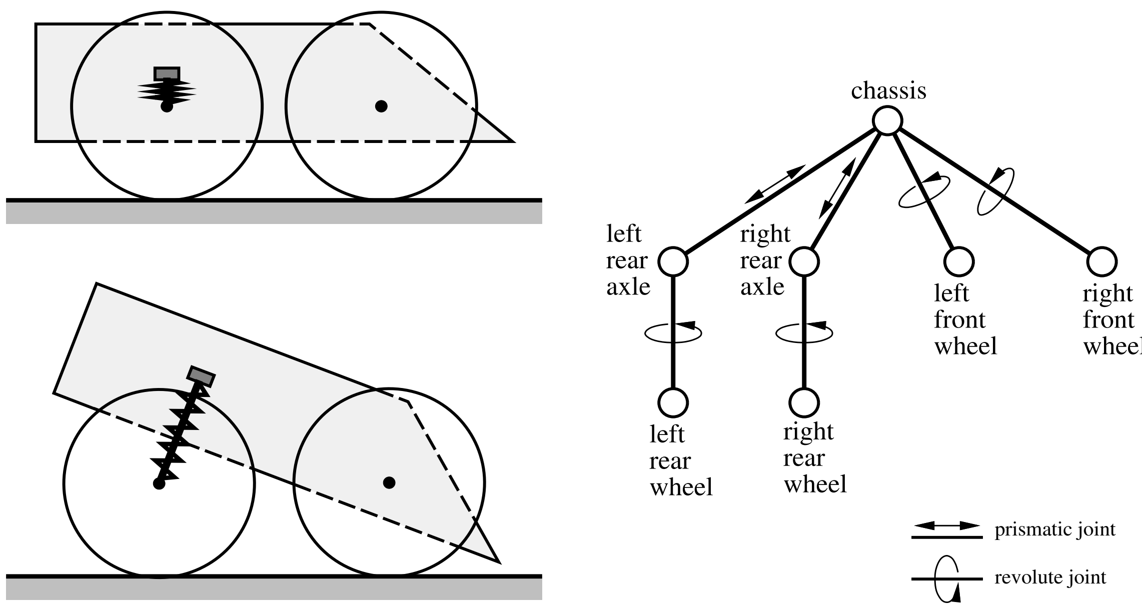

Figure 7.25: Left: *Side view of the rover in normal and "jacked" configurations. Jacking allows the rover to push bugs rather than running over them.* Right: *Kinematic tree for the rover, showing the individual links and joints.*

different desired velocities to the left and right side wheels. Two additional PD controllers regulate the length of the rear prismatic pistons. By activating these controllers, the rover can jack up its rear end and lower its front end in preparation to push bugs. Deactivating these controllers makes the rover resume its normal driving posture. The rover's behavior is more complicated than that of the bugs. It repeatedly chooses one of the surviving bugs as a victim, chases this bug down, assumes jacked position, and pushes the bug off the edge of world and into the abyss. It avoids driving itself over the edge. This behavior was implemented via a state machine controller with five states, shown in Figure 7.26.

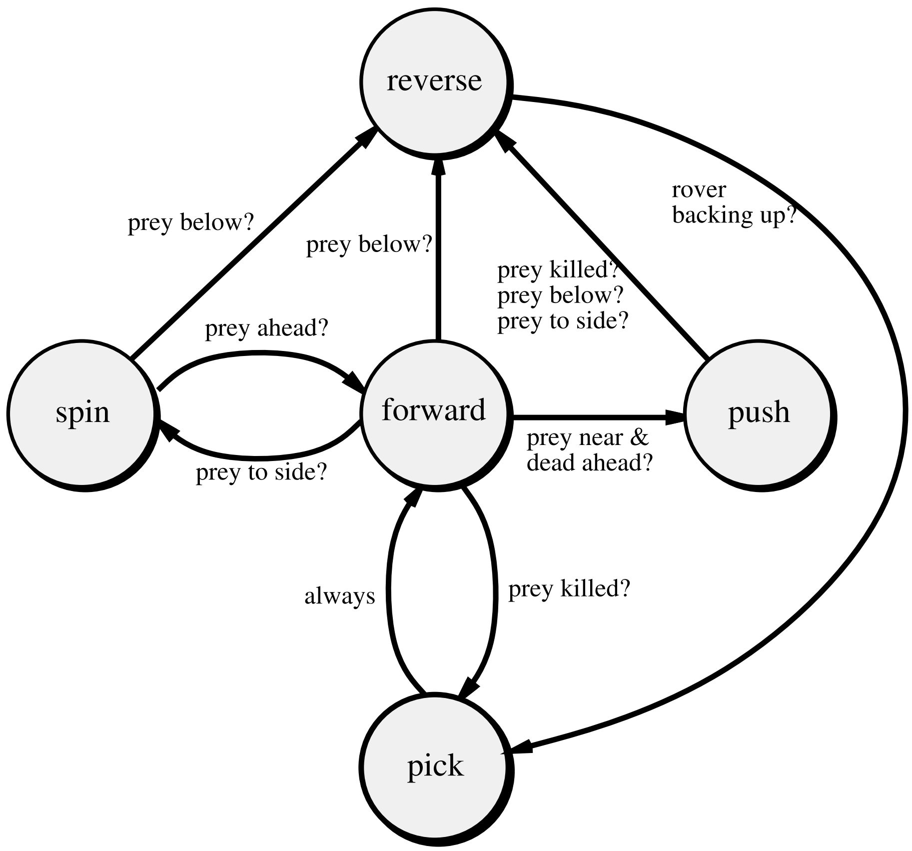

Figure 7.26: *The high level state machine controller for the rover. There are five states, and the edges are labeled with the condition for the corresponding state transition.*

The PD controller rates for this example vary from 1 to 2 kHz, and the high level control routines run at 100 Hz. Two high level control routines are required for this animation, one for the bugs and one for the rover. Both are C functions, about one page long. The actual function for implementing the rover is shown in Figure 7.27. The *creature war* shows how interesting behaviors can emerge from physics plus simple control laws and interactions between agents. The sequence of the final animation was quite unimagined when the control laws were designed.

```
void cli_rover(MB *mb)
{
    FILE *fp;
    REAL Kp, Kd, Ki, period, desired;
    REAL Kp2, Kd2, Ki2, period2;
    REAL Ks, Ls, damping;

    fp = fopen("rover.ctrl", "r");
    fscanf(fp, "%lf %lf %lf %lf %lf", &Kp, &Kd, &Ki, &period, &roverVmax);
    fscanf(fp, "%lf %lf %lf %lf", &Kp2, &Kd2, &Ki2, &period2);
    fscanf(fp, "%lf %lf %lf", &Ks, &Ls, &damping);
    fclose(fp);

    installSpring (&mb->joint[1], Ks, Ls);
    installSpring (&mb->joint[2], Ks, Ls);
    installDamper (&mb->joint[1], damping);
    installDamper (&mb->joint[2], damping);

    createPID (mb, 1, period2, Kp2, Kd2, Ki2, 0);
    createPID (mb, 2, period2, Kp2, Kd2, Ki2, 0);
    createPID (mb, 3, period, Kp, Kd, Ki, 1);
    createPID (mb, 4, period, Kp, Kd, Ki, 1);
    createPID (mb, 5, period, Kp, Kd, Ki, 1);
    createPID (mb, 6, period, Kp, Kd, Ki, 1);

    roverPrey = 0;
    roverState = ROVER_FWD;
    createGenCtrl (mb, (REAL) 0.01, roverCtrl);
}
```

```
void roverCtrl(CTRL_LAW *cl)
{
    MB *mb;
    RB *chassis, *prey;
    REAL vLeftDes, vRightDes, d, bearing, deltaV, k;
    REAL deltaBear, spin, inc;
    int dead, i;
    VECT3 goal, v;
    JOINT *jnt;
    SE3 Tib;

    static char *states[ ] = {"fwd ", "spin", "push", "rev ", "pick"};

    mb = cl->body;
    chassis = mb->link[0];
    prey = MBtable[CRAWLER1 + roverPrey].link[0];
    se3inv(&chassis->Tbi, &Tib);
    if (roverPrey >= NUM_CRAWLERS) {v[0] = v[1] = 0.0; v[2] = 5.0;}
    else vectCopy(prey->Tbi.d, v);
    dead = (v[2] < 0.0);

    se3xformPoint(&Tib, v, goal);
    d = sqrt(SQR(goal[0]) + SQR(goal[1]));
    vectNormalize(goal, goal);

    inc = 90 - acos(goal[2]) * RAD_TO_DEG;
    bearing = atan2(goal[1], goal[0]) * RAD_TO_DEG;
    deltaBear = fabs(bearing);
    se3xformVect(&chassis->Tbi, chassis->w, v);
    spin = v[2];

    switch(roverState) {

    case ROVER_FWD:
        deltaV = roverVmax * (0.04 * bearing - 0.15 * spin);
        vRightDes = roverVmax + deltaV;
        vLeftDes = roverVmax - deltaV;
        if (dead) roverState = ROVER_PICK;
        else if (inc < -20) roverState = ROVER_REV;
        else if (deltaBear > 60) roverState = ROVER_SPIN;
        else if (d < 50.0 && deltaBear < 25) {
            setPID (mb, 1, 6.0);
            setPID (mb, 2, 6.0);
            roverState = ROVER_PUSH;
        }
        break;

    case ROVER_SPIN:
        vRightDes = roverVmax * (0.04 * bearing - 0.15 * spin);
        vLeftDes = - vRightDes;
        if (inc < -20) roverState = ROVER_REV;
        else if (deltaBear < 20) roverState = ROVER_FWD;
        break;

    case ROVER_PUSH:
        if (roverPrey >= NUM_CRAWLERS) {roverState = ROVER_REV; break;}
        deltaV = roverVmax * (0.04 * bearing - 0.15 * spin);
        vRightDes = roverVmax + deltaV;
        vLeftDes = roverVmax - deltaV;
        vectCopy(prey->v, v);
        k = (SQR(v[0]) + SQR(v[1])) / 300.0;
        displacePoint(prey->Tbi.d, v, k / vectNorm(v), v);
        if (v[0] < -200 || v[0] > 200 || v[1] < -200 || v[1] > 200
            || deltaBear > 25 || inc < -20) roverState = ROVER_REV;
        break;

    case ROVER_REV:
        vLeftDes = vRightDes = - 2.0 * roverVmax;
        se3xformVect(&Tib, chassis->v, v);
        if (v[0] < -5.0 && inc > -10) roverState = ROVER_PICK;
        break;

    case ROVER_PICK:
        vLeftDes = vRightDes = 0.0;
        if (dead) roverPrey++;
        disablePID (mb, 1);
        disablePID (mb, 2);
        roverState = ROVER_FWD;
        break;
    }

    vLeftDes /= ROVER_WHEEL_RAD;
    vRightDes /= ROVER_WHEEL_RAD;

    setPID (mb, 3, vLeftDes);
    setPID (mb, 4, vLeftDes);
    setPID (mb, 5, vRightDes);
    setPID (mb, 6, vRightDes);
}
```

Figure 7.27: *The C functions that implement the rover's high level control. Function* `cli_rover` *is called once to instantiate controllers and set initial values. Function* `roverCtrl` *implements the high level state machine depicted in Figure 7.26. It is called by the simulator scheduler every 0.01 seconds. Calls to the control library are in underlined and in bold font.*

7.4 Execution times

Table 7.1 summarizes the execution times needed to simulate the examples described above. *Simulation time* is the length of time which passed in the simulation, *com-*

example	simulation time (sec)	computation time (sec / min:sec)	slowdown
pure impulse-based simulation			
dominos	1.2	3.7	3.1
block drop	0.8	0.4	0.5
block on ramp	5.0	1.4	0.3
chain of balls	2.5	3.0	1.2
balls in dish	5.8	14.0	2.4
coins	3.6	16.7	4.6
pool break	3.0	27.0	9.0
bowling	5.0	18.0	3.6
ball on spinning platter	40	54.3	1.4
rattleback top	10.0	12.0	1.2
part feeder chute	5.0	33.0	6.6
exploding text	4.0	18.0	4.5
passive hybrid simulation			
see-saw	7.0	22.3	3.2
part sorter	2.5	53.7	21.5
triple pendulum	6.0	2:35	25.9
controlled hybrid simulation			
furniture arranging	35.9	31:24	53
bicycle	38.9	20:08	31
creature war	21.2	14:04	40

Table 7.1: *Simulation times for examples.*

putation time is the actual time needed to compute the simulation, and *slowdown* is the ratio of the latter to the former (a slowdown of 1.0 corresponds to real time simulation). Simulations were performed on an *SGI Indigo II* with 200 MHz R4400 processor. Execution times shown are the average taken over several runs, with graphics turned off. For the pure impulse-based simulation examples, running times were generally less than five times slower than real time; for a couple of the examples, real time simulation was achieved.

Impulse ran slower on the hybrid examples, generally one to two orders of magnitude slower than real time. The slowdown in the *part sorter* and *furniture arranging* examples was primarily due to collision detection involving several highly nonconvex bodies. For the *triple pendulum*, *bicycle*, and *creature war* examples, the bulk of the processing time was spent computing nontrivial multibody forward dynamics. The *see-saw* example, which had only convex objects and a very simple multibody, ran relatively quickly.

Table 7.2 details how the processing time was spent for the three controlled hybrid simulation examples. In all cases, only a very small fraction of time was spent executing

example arrangement	forward dynamics	collision detection	collision response	control system
furniture	4.0	92.6	3.3	0.1
bicycle	80.6	15.5	3.5	0.5
creature war	72.1	19.3	3.2	5.4

Table 7.2: *How the simulation processing time is spent. All figures are percentages of total processing time.*

control laws. Collision response, the time spent computing and propagating multibody collision impulses, is also fairly small. As mentioned before, the nonconvex collision detection uses most of the processing time in the *furniture arranging* example, and computing forward dynamics uses most of the time in the other examples. Significant speed up is possible by using commercial packages that precompile the dynamics computation for greater efficiency (as in [HWBO95]). Also, under impulse-based simulation, the dynamics of separate multibodies may be computed independently, so parallelization is trivial.

For rigid body and multibody forward dynamics integration, as well as for collision integration, *Impulse* uses a standard fourth order Runge-Kutta integrator, with adaptive step size [PTVF92]. To take one step forward, the integrator calls the derivative evaluation routine 11 times. This can be quite expensive compared to, say, a non-adaptive Euler integration step, which only requires one derivative evaluation per step. An impulse-based simulator tends to call the dynamics integrator with fairly small intervals, since the time between collisions is short, and with these small intervals the accuracy of a simpler and faster integrator might be sufficient. Further study is needed.

7.5 Estimating pose statistics

The initial problem that sparked the development of *Impulse* was the simulation of vibrational parts feeders. These machines, commonly used in automated assembly, use high frequency, small amplitude vibratory motion to move and orient parts, which are often subsequently handled by a manipulator. More detailed descriptions of these types of feeders may be found in [BPM82]. Vibrational parts feeders naturally led to the idea of modeling contact with impulses, since the contact modes between the part and feeder are highly transient. Many non-vibrational feeders are natural applications for impulse-based simulation, because the parts still collide frequently with the feeder, and collisions have a dominant effect on the dynamics of the parts.

Impulse was used to predict the probabilities of rigid parts landing in their various stable poses, when dropped onto a flat surface. The problem has specific relevance to the Adept Robotics flexible feeder, but would also be useful for other feeders. Results from the simulator and from quasi-static methods were compared to data from physical experiments for a number of real industrial parts; these results are included below. More details of the methods used to estimate pose statistics are in [MZG+96].

7.5.1 Background

In contrast to fixed assembly lines, flexible assembly lines can be rapidly reconfigured to handle new parts. This can dramatically reduce the time needed to bring new products to market and permit the cost of assembly lines to be amortized over multiple products. *Part feeders*, which singulate and orient the parts prior to packing and insertion, are critical components of the assembly line and one of the biggest obstacles to flexible assembly. Currently, the design of part feeders is a black art that is responsible for up to 30% of the cost and 50% of workcell failures [NW78, BPM82].

Carlisle *et. al.* [RG93] proposed a flexible part feeding system that combines machine vision with a high-speed robot arm. In contrast to custom-designed hardware such as the bowl feeder, only software is changed when a new part is to be fed. The idea is that a collection of like parts are randomly scattered on a flat worktable by the force of gravity. An overhead vision system determines the pose (position and orientation) of each part. The robot arm then picks up each part and moves it into a desired final pose as illustrated in Figure 7.28. A method of estimating the throughput of this feeder is in [GC95], but the method requires knowledge about the distribution of poses, particularly orientation, when the parts are scattered on the flat worktable. More formally, the problem to be solved is:

Problem 10 (Estimating Pose Statistics (EPS)) *For a rigid polyhedral part X with known center of mass and inertia tensor, denote the n faces of its convex hull H by $F_1, \ldots, F_n$. Assuming X is repeatedly dropped onto a flat surface from some known distribution of initial poses, compute the values $p_1, \ldots, p_n$, where p_i is the probability that X reaches a final resting state with F_i down (against the surface).*

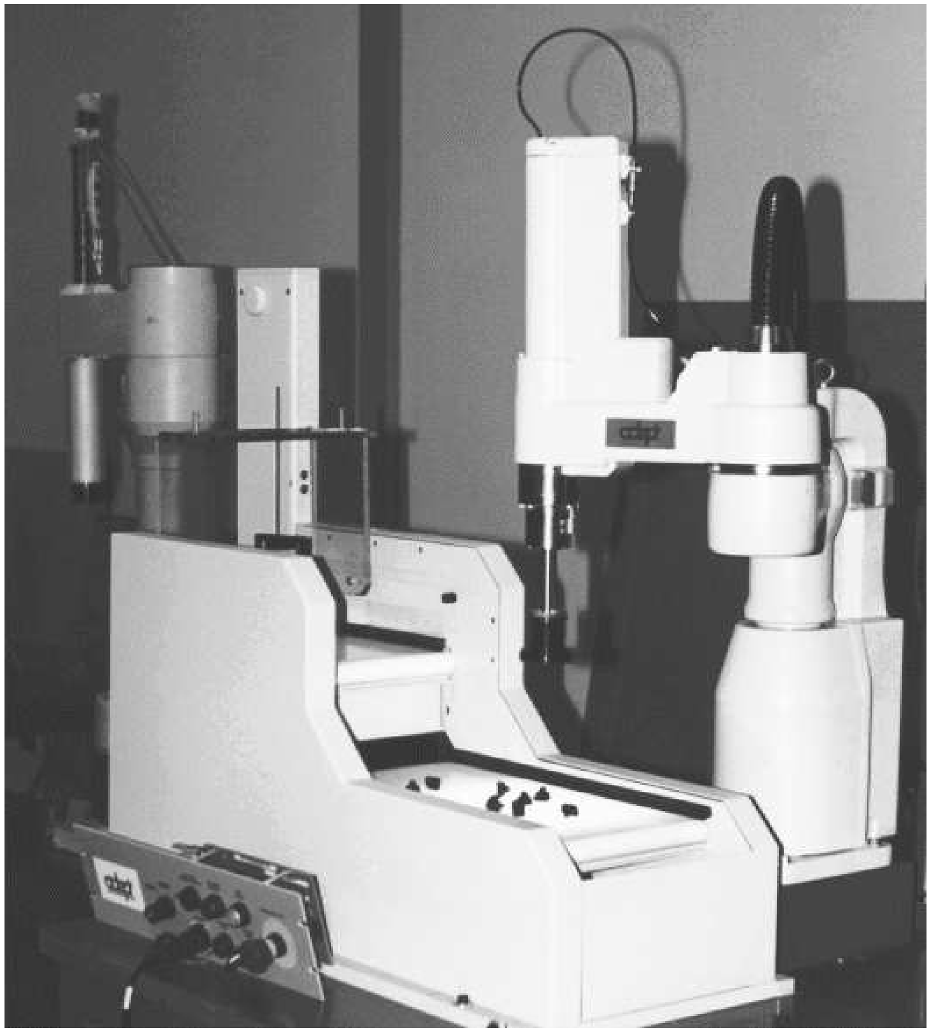

Figure 7.28: *A flexible part feeder using machine vision, a high-speed robot arm, and pivoting gripper. This illustration shows the system feeding car-stereo pushbuttons.*

Additional assumptions are that the workspace is flat and much larger than X, and that X does not collide with other parts.

Predicting the pose distribution of rigid parts dropped onto a flat surface is important in evaluating part designs for assembly, in estimating feeder throughput, and in determining how many robots and assembly lines are required to meet specified production rates. This work in estimating pose statistics complements other ongoing work in automated assembly. Rao, Kriegman, and Goldberg have studied the use of a pivoting gripper for Adept's flexible feeder; they give an $O(m^2 n \log n)$ algorithm to generate pivot grasps for a part with n faces and m stable configurations [RKG95]. Christiansen, Edwards, and

Coello Coello give a genetic algorithm for designing efficient part feeders from component gates [CEC96]. Their algorithm takes pose statistics, such as the ones computable by *Impulse*, as input.

7.5.2 Quasi-static algorithms

For completeness, Problem 10 was also solved using two quasi-static algorithms. The first algorithm was proposed by Wiegley *et. al.* [WRG92], and is an improvement on the early work done by Boothroyd [BPM82]. Motion of the part is assumed to be quasi-static; inertia and velocity are neglected.

The idea of the algorithm is illustrated in Figure 7.29. After computing the part's

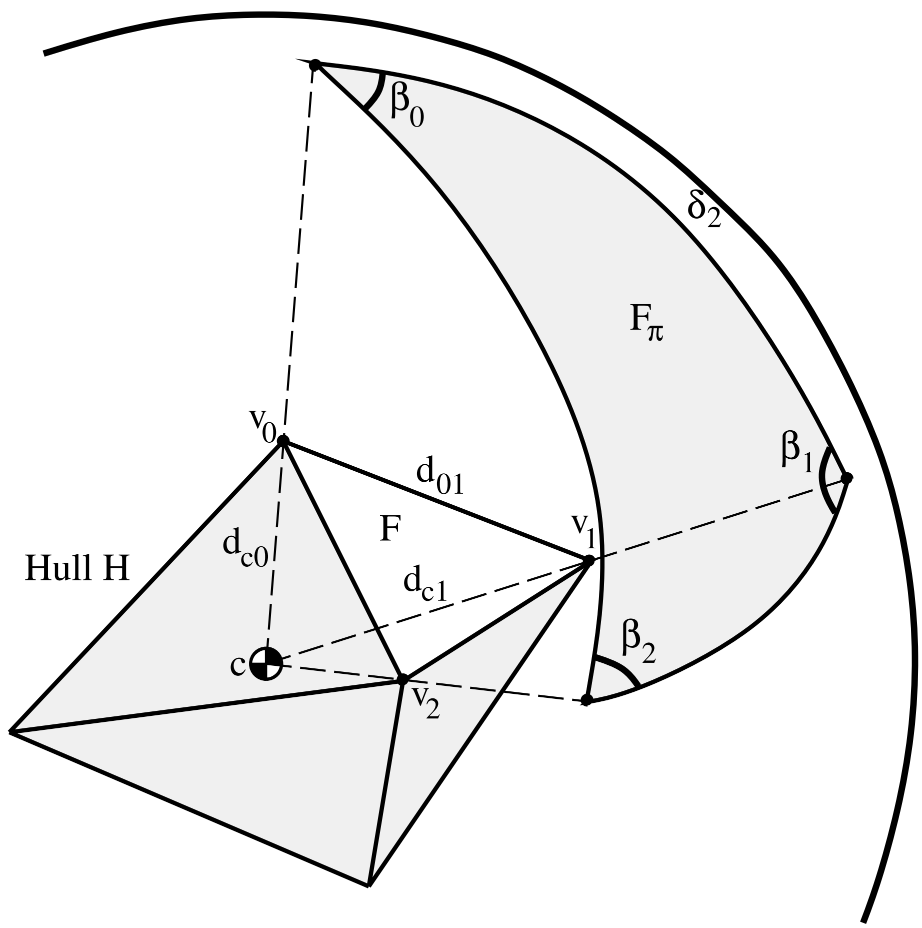

Figure 7.29: *Estimating pose statistics using a sphere projection method.*

convex hull H, the facets of H are projected onto a unit sphere centered at the center of mass c. If F_π is the projection of face F, then the ratio of the area of F_π to the total surface area of the sphere gives the probability that the part will land on face F under quasi-static conditions, if its initial pose is uniformly distributed over $SO(3)$.

Assuming triangular faces, the ratio in question is given by

$$A = \frac{\beta_0 + \beta_1 + \beta_2 - \pi}{4\pi}, \tag{7.1}$$

where the β_i are the interior angles of F_π; the β_i can be computed from standard spherical trigonometry.

This procedure results in an initial estimate of each p_i. To treat faces of H that are statically instable, the center of mass is projected onto the plane of each face F_i. If the projected point lies outside face F_i, gravity will cause the part to topple over to adjacent face F_j. In this case p_i is added to p_j, and p_i is set to zero. The algorithm to compute the pose statistics is $O(n^2)$ in the number of polyhedral edges.

Since the quasi-static analysis does not model dynamic disturbances, it often overestimates the probability of landing on a facet that is stable but easily dislodged by small vibration. Consider two facets of the part's convex hull, F_i and F_j, and the bounding edge e between them. Let g be the gravity vector from the part's center of mass. If g intersects F_i when in contact, the part will remain on facet F_i, under the quasi-static model. However, dynamic energy may rotate across edge e when g points inside facet F_i but is close to edge e (Figure 7.30). A second quasi-static method, called the perturbed quasi-static method, computes a more accurate likelihood that the part might rotate from facet F_i to facet F_j, and propagates probabilities accordingly (see [MZG$^+$96] for details). In all the experiments, the cone half-angle was 20°, although the right way to choose this half-angle measure is not completely clear.

7.5.3 Dynamic simulation

The EPS problem was attacked using full dynamic simulation of the parts being dropped onto a surface, using *Impulse*. For dynamic simulation, many parameters not used in the quasi-static and perturbed quasi-static algorithms are important. The coefficients of friction and restitution were both estimated to be 0.3.[2] The feeder configuration is also

[2]Recently, Issa Nessnas of Adept has experimentally measured these coefficients. He estimated the coefficient of friction to be 0.8–1.4, and the coefficient of restitution to be 0.17–0.31, although there was a large amount of noise in these results.

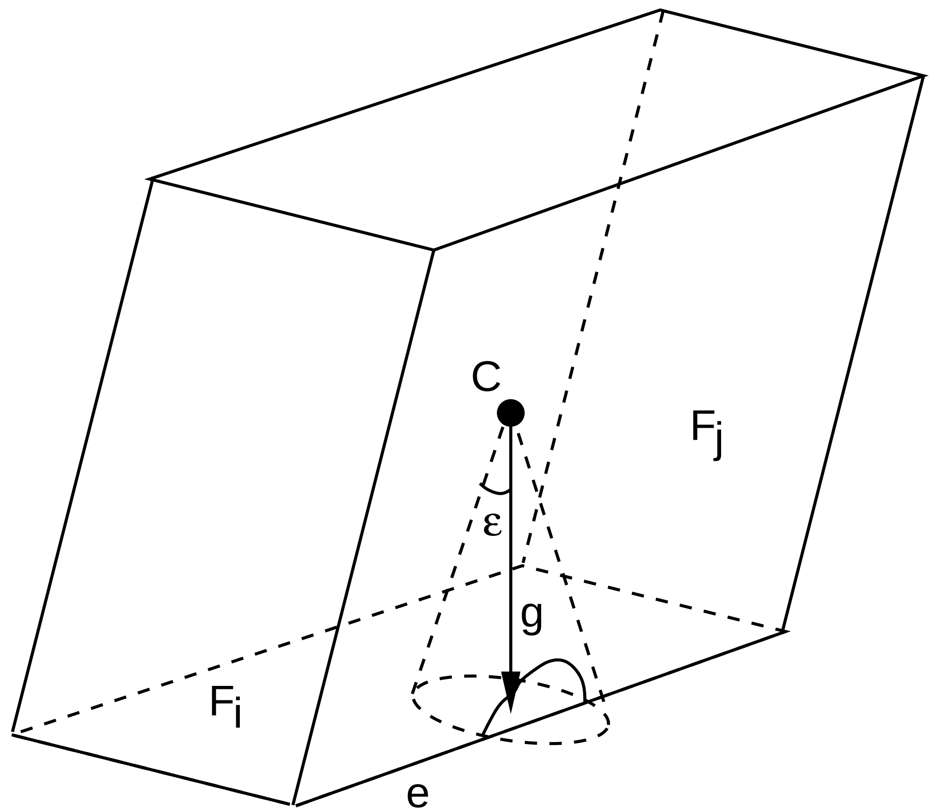

Figure 7.30: *The perturbed quasi-static method uses a cone projected down from the center of mass, to estimate the likelihood that the part will topple from face F_i to face F_j. This likelihood is used to propagate some of the probability from p_i to p_j.*

important. Adept's flexible feeder system dumps the part from an upper belt onto a lower belt, where the parts are examined by the vision system, and then picked and placed by one or more manipulators (Figure 7.28). For dynamic simulation, the height of the drop was estimated at 12.0 cm. The horizontal velocity of the parts as they leave the upper belt was estimated at 5.0 cm/s.

The initial orientation of the parts poses a problem since the parts are in a stable resting state on the upper belt, before being dropped onto the lower one. Thus, the initial distribution of orientation is not uniform, but similar to the final (initially unknown) distribution. To circumvent this problem, the initial orientations for the first 20 drops were chosen randomly, assuming a uniform distribution over $SO(3)$. This bootstraps the process with a preliminary final pose distribution. For all remaining drops, the initial (upper belt) poses are chosen from the current distribution of final (lower belt) poses, so that the results

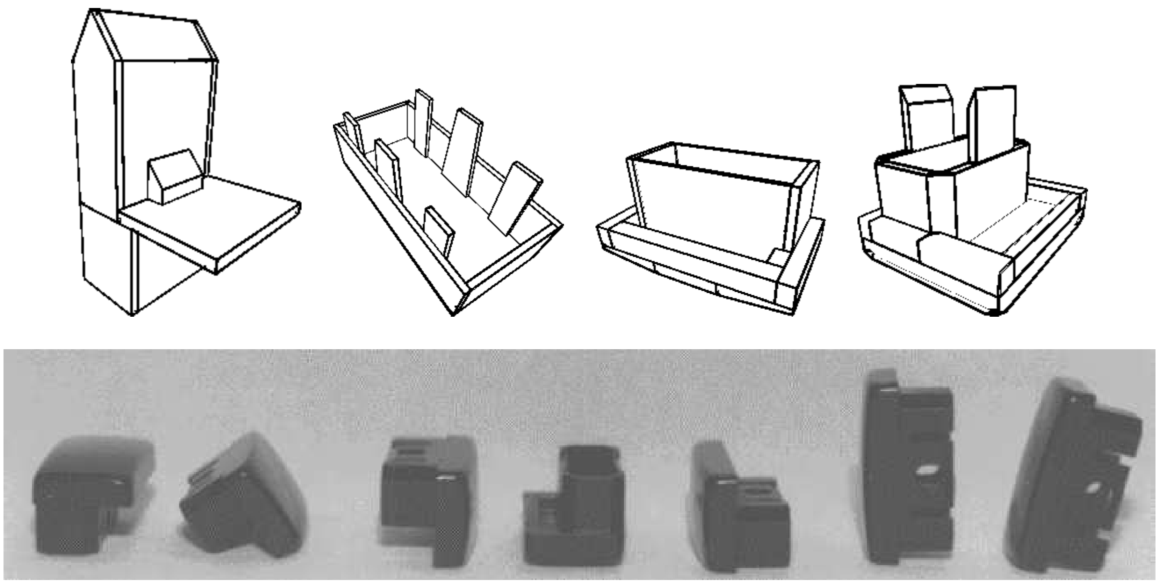

Figure 7.31: Top: *CAD models of the four parts used in the experiments. From left to right: insulator cap, large white, rectangular black, and square black stereo buttons.* Bottom: *Photographs of the rectangular black stereo button in its seven stable states.*

of the drop tests are continually fed back to determine initial conditions. A slight perturbation (a rotation of up to 1.5 degrees about a randomly chosen axis) is also applied to the initial pose to introduce noise into the system due to belt vibration.

7.5.4 Experimental results and discussion

The three algorithms described were applied to four test parts. The test parts were all small, plastic, rigid pieces, of the type typically used in automated assembly (Figure 7.31). Part #1 is an insulator cap purchased at a local hardware store. Parts #2, #3, and #4 are pushbuttons designed for a commercial car stereo system. Geometric models of each part were constructed by measuring the parts with a ruler. Centers of mass and moments of inertia for the parts were computed automatically by *Impulse*, using the techniques of Chapter 6. As a control, physical experiments were performed by repeatedly running several hundreds of samples of each test part through Adept's flexible feeder system (Figure 7.28); the final resting poses were recorded by a human observer. Tables 7.3 through 7.6 show the results. All quantities in the tables are percentages.

The error percentages included in the tables indicate the overall performance of each algorithm for each sample part. They are computed as the average deviation of the algorithm's predictions from the physical test percentage, weighted by the frequency with which that state actually occurs. Let $p_1, \ldots, p_n$ represent the probability of each of n states, as measured in the physical test. Let $a_1, \ldots, a_n$ represent the corresponding probabilities computed by one of the algorithms. The error percentage for that algorithm is given by

$$e = 100 \sum_{i=1}^{n} p_i \; |a_i - p_i|. \tag{7.2}$$

Pose	Quasi-Static	P'turbed Q–S	Dynamic Sim.	Physical Tests
1	30.5	46.5	41.9	46.0
2	37.3	30.2	26.2	27.1
3	19.6	19.8	28.3	19.7
4	8.3	3.5	3.0	5.0
5	4.2	0.0	0.8	2.2
error	10.1	1.2	4.0	–

1036 trials performed in physical experiments.

Table 7.3: *Orange insulator cap data.*

Pose	Quasi-Static	P'turbed Q–S	Dynamic Sim.	Physical Tests
1	34.5	48.8	71.7	75.8
2	39.9	30.6	20.9	13.8
3	19.2	20.5	7.4	10.5
4	6.3	0.0	0.1	0.0
error	35.8	23.8	4.4	–

545 trials performed in physical experiments.

Table 7.4: *White stereo button data.*

The quasi-static and perturbed quasi-static algorithms are extremely fast, requiring less than a second of computation time for parts with 50 facets, and the perturbed quasi-static algorithm's predictions are consistently more accurate than those of the quasi-static one. Dynamic simulation is slower. For each part, 2000 drops were simulated, taking approximately 45 minutes per part, however greater accuracy is also achieved. The dynamic simulation algorithm is the most accurate for all sample parts, except the insulator

Pose	Quasi-Static	P'turbed Q–S	Dynamic Sim.	Physical Tests
1	36.2	47.3	54.1	56.0
2	16.0	25.5	24.1	24.5
3	17.4	17.0	14.0	13.6
4	8.1	1.2	1.4	4.4
5	10.6	4.5	5.3	1.4
6	7.5	4.4	1.0	0.3
7	4.3	0.0	0.3	0.0
error	14.0	5.8	1.4	–

1099 trials performed in physical experiments.

Table 7.5: *Rectangular black stereo button data.*

Pose	Quasi-Static	P'turbed Q–S	Dynamic Sim.	Physical Tests
1	35.7	46.6	68.4	62.2
2	17.5	15.5	16.6	15.2
3	12.1	17.0	6.1	11.0
4	7.2	8.6	6.0	4.7
5	3.9	1.6	2.7	3.1
6	5.6	1.5	0.0	2.8
7	3.8	3.9	0.3	0.5
8	4.2	1.7	0.0	0.0
9	3.0	2.3	0.0	0.0
10	2.6	0.7	0.0	0.0
11	2.2	0.0	0.0	0.0
12	2.1	0.5	0.0	0.0
error	17.2	10.7	4.8	–

915 trials performed in physical experiments.

Table 7.6: *Square black stereo button data.*

cap (Table 7.3), for which the perturbed quasi-static algorithm slightly outperforms it. The dynamic simulation algorithm's prediction accuracy is also the most consistent; the composite error is less than 5% in all cases.

In an interactive setting, where a designer is perhaps editing the CAD model of a part in order to improve feeder throughput, the perturbed quasi-static algorithm is clearly the best choice. The designer need only wait seconds to see how changing a part's CAD model alters the pose distribution and feeder throughput. The dynamic simulation algorithm is useful for obtaining a more accurate estimate once the design has been determined, or for analyzing the effects of more subtle design changes. It models several factors, that aren't considered by the standard and perturbed quasi-static algorithms, including: friction, collisions with energy loss, mass moments of inertia, height of drop, and initial conditions of the part prior to drop. To study the effects of varying these parameters, dynamic simulation is appropriate.

Our simulation experiments involved 2000 drops tests. Often, fewer trials may be sufficient, reducing the computational cost of this method. Suppose the true (unknown) probability that a part lands in a particular pose is p. The number of times the part lands in this pose over n trials is a binomial random variable, which may often be approximated by a normal distribution.[3] A *confidence interval* statement is of the form: "p lies within the range $(\mu-\delta, \mu+\delta)$, with $100(1-\alpha)\%$ certainty." Here, μ is the probability estimate obtained from the n trials, δ is the allowable error tolerance, and α is the *level* of the statistical test. Given δ and α, one can bound the number of trials necessary by

$$n = \frac{\Phi^{-1}(1 - \frac{\alpha}{2})}{2\delta}, \tag{7.3}$$

where $\Phi(x)$ is the cumulative normal distribution function. For example, to pinpoint the probability of a particular final pose to within 5%, with 90% certainty, $\delta = 0.05$ and $\alpha = 0.10$. From (7.3), 385 trials are sufficient. See [Dev82] for more information.

7.5.5 Other part feeding experiments

Impulse is currently being used for several other studies in part feeding. Impulse-based simulation is a good match to this domain, because the interactions between feeder and parts are often collision intensive, or of a vibratory nature. An accurate friction model is also required for accurately predicting the behavior of many feeders.

Berkowitz and Canny have developed a tool for designing industrial part feeders based on stochastic data from dynamic simulation [BC96]. In many feeders, parts are propelled along a feeder track, through vibration, conveyor belt, or gravity. Various protrusions and cut-outs (also called *gates*) along the path reorient the passing parts, or reject them; parts passing successfully through all of the gates are correctly oriented. Feeder designs can be parameterized by the positions, angles, and dimensions of the gates. Berkowitz's

[3] A common rule of thumb is that the normal approximation is valid if the numbers of successes and failures during the trial series both exceed five.

and Canny's tool allows a designer to search through the space of possible feeder designs, searching for the optimal one with respect to a metric the designer specifies. *Impulse* is used to compute the trajectories of the parts through the feeders; the impulse-based approach is very efficient for this type of simulation.

Another way to apply this idea would be for the algorithm of Christiansen *et. al.* They describe an approach to near-optimal design of vibratory bowl feeders, using a genetic algorithm to search through the space of all gate sequences [CEC96]. For each possible gate, their algorithm requires as input a stochastic matrix that characterizes the gate operation: element (i, j) of the matrix is the probability that a part entering a gate in state i will exit in state j. Impulse-based simulation can easily gather this type of information. A similar application is the algorithm of Wiegley *et. al.* that generates a sequence of passive fences to align parts moving on a conveyor belt. In [WGPB96], they analyze their feeders using a frictionless, quasi-static model.

A completely different type of part feeding involves micro-actuated motion arrays, a type of micro electro-mechanical (MEM) device. These devices also have applications in video displays. Böhringer *et. al.* have fabricated a motion array comprising several thousand torsional resonators distributed over a surface of a few square centimeters [BDMM94], however difficulties in producing and testing these motion arrays have hindered experimental results. Reznik and Brown are using *Impulse* to analyze motion arrays based on the model of Böhringer *et. al.*. Their study focuses on determining feed rates for the arrays, and how the performance is affected by variations in certain design parameters and operating conditions [RB]. Results such as these might increase understanding of these devices and suggest future improvements. Figure 7.32 depicts one of Reznik's and Brown's experiments. Electrostatic stepper motors are another type of MEM device that could be improved with results from simulation. Since their operation depends on transient contact between components, analytic results are difficult to derive. Zhuang is using *Impulse* to analyze MEM stepper motors and the effects of varying design parameters [Zhu96].

Acknowledgments. Thanks to Neil Getz for providing his bicycle controller and explaining its operation, and to Jeff Wendlandt for many enlightening discussions. Section 7.5 was largely extracted from [MZG$^+$96]; thanks to Randy Brost and Bruce Shimano for useful feedback on the initial draft of that paper. Versions of the quasi-static algorithm of

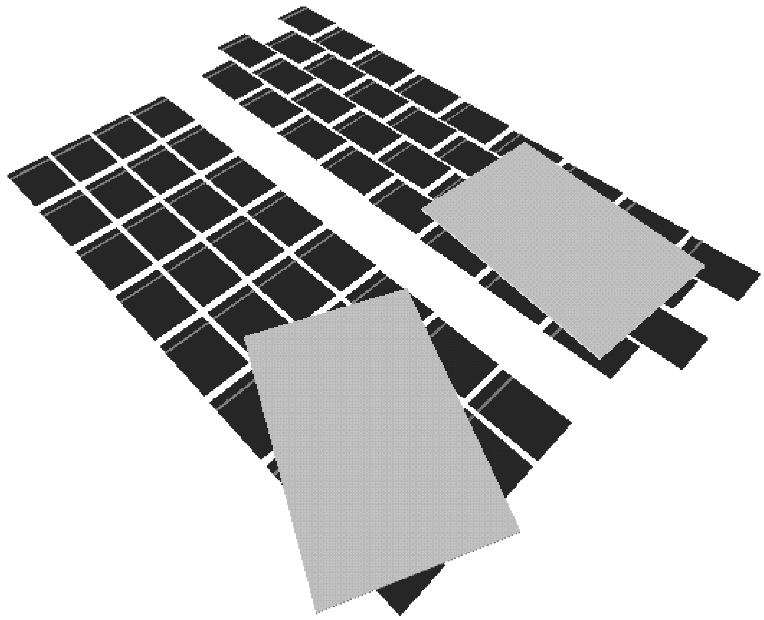

Figure 7.32: *A snapshot taken from* Impulse *during one of the MEM motion array experiments described in [RB]. Here, two different resonator tilings are compared to determine possible differences in feed rates (reprinted with permission).*

Section 7.5.2 were implemented at USC: Wiegley in 1992, Zheng Yeh in 1993, and Yan Zhuang in 1994. The convex hull routine is due to Ioannis Emiris and John Canny. Special thanks to Stephen Chenney for the renderings of the controlled hybrid simulations, shown in Figures 7.18, 7.19, and 7.23.

Chapter 8

Future Work

This thesis has introduced impulse-based simulation, a paradigm for dynamic simulation of rigid body and articulated body systems. The approach, pioneered by Hahn, has been extended, refined, and implemented with the prototype simulator *Impulse*. Tests with *Impulse* show that the method is capable of interactive simulation speeds for moderately complex physical systems, and that the physical accuracy of the method is suitable for many applications. This final chapter mentions some challenges that remain and some avenues for future research.

8.1 Paradigm switching

Hybrid simulation, as presented in this thesis, is a somewhat limited blending of constraint- and impulse-based simulation techniques. Constraints hold together the joints of a multibody, while an impulse-based approach is used for the contacts between different multibodies. This expands the domain of what can be done using impulse-based approaches, but fails to address some basic deficiencies in the method. What can be done in the case of a stack of books on a table? Here, a constrained model for contact between bodies is appropriate, but the presented hybrid methods do not employ one. One way to solve this problem would be to detect the onset of static contact between bodies. This might be done by observing that a high number of low velocity impacts are occurring between adjacent books in the stack. At this point, a *paradigm shift* could be made to model the contact interactions between books with constraint forces.

The transition need not be a sharp one. A contact force f' that is less than the true contact force f might initially be applied between the books; the slack would be picked up by collisions. Even the force f' would reduce the number of collisions and increase the simulation speed. As f' approaches f, the number of collisions decreases to zero, and pure constraint-based modeling is achieved. A gradual paradigm shift like this seems less susceptible to anomalous behavior that might result from discontinuities in simulation strategy. This impulse- to constraint-based shift could naturally and efficiently handle a very common case: objects coming into initial contact through collision, followed by some period of low impact collisions as the objects settle into continuous contact.

Constraint- to impulse-based paradigm shifts are also appropriate. Consider: an arrangement of bowling pins in resting contact with the ground until struck by the inevitable ball; a stack of blocks at rest until the bottom one is suddenly removed; or a skateboard in steady rolling contact with the ground until a speed bump sends it airborne. In all of these situations, a switch from a constraint-based to an impulse-based approach seems appropriate in preparation for periods of transient, collision intensive contact. An abrupt change in the normal force applied between a pair of bodies might be the triggering event for this shift from constraint-based to impulse-based modeling of the contact.

We believe that a fuller blending of the impulse- and constraint-based approaches through paradigm shifts is a very fruitful area for future research. By applying each technique in the cases to which it is well suited, the range of physical systems that can be efficiently simulated can be significantly increased.

8.2 Collision detection issues

Of major influence in the design of *Impulse's* narrow-phase collision detection system was the fact that the Lin-Canny algorithm does not operate correctly on intersecting polyhedra. This motivated the one-sided approach to finding the moment of collision. In the more common two-sided approach, the simulation regularly proceeds past the moment of collision, at which point the collision detection system detects penetration and the simulation is backed up. Binary search or similar techniques can pinpoint the moment of collision to within a prescribed spatial or temporal tolerance. A disadvantage to the two-sided approach is that it is susceptible to missing collisions. But the two-sided approach is likely to converge to the collision time faster than the one-sided approach, which requires conservative lower bounds on the time of impact. The two-sided approach is certainly easier to implement when the objects have complex dynamics, making conservative bounds hard to obtain. If a way to avoid missing collisions was devised, a two-sided approach might be more efficient for impulse-based simulation.

Parallelization is an important advantage of the impulse-based method, but it extends only to the dynamic integration phase. Parallel collision detection algorithms could greatly improve performance on parallel machines. Using the single collision heap scheme described in this thesis, whenever a collision check occurs between any pair of bodies, the dynamic integration of every body stops. This is clearly inefficient. If the bodies were partitioned into separate heaps, based on spatial locality, the dynamic integration of a body would not be halted by irrelevant collision check events; the individual heaps could be parceled among the different processors. This partitioning is advantageous in single processor settings as well, since certain bodies could be evolved further in time for each call to the integrator, resulting in larger integration steps and faster simulation. The situation would be as depicted in Figure 8.1. All body integrations need not stop for every collision

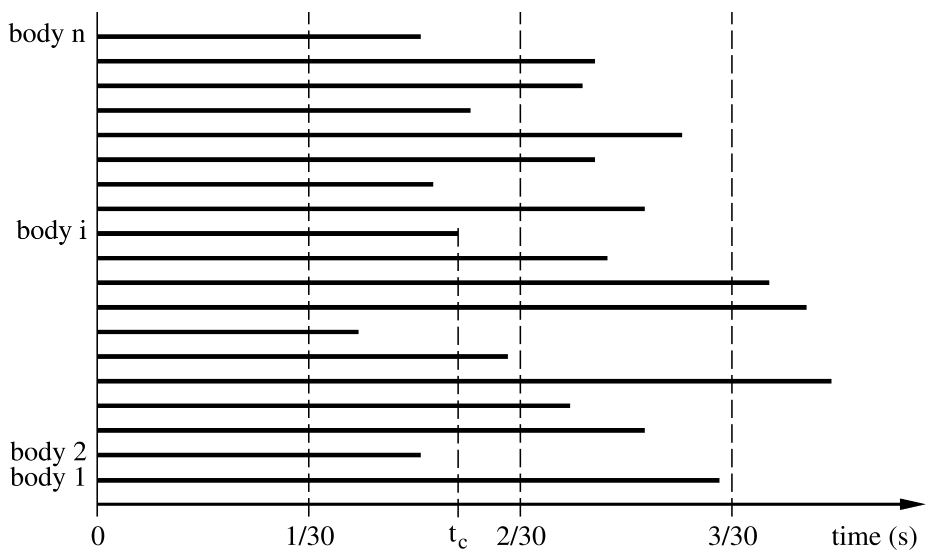

Figure 8.1: *If the dynamic integrations of every body are not required to stop for every collision check, then the evolution of the states does not have to proceed in lockstep.*

check, so the evolution of the bodies does not proceed in lockstep. The bars in the figure indicate the time to which each body's state has been evolved; for example, body 1 has been evolved to almost 3/30 seconds. Whenever all bodies have been evolved to the next frame refresh time, the screen is redrawn with their positions at that time. Some consistency problems must be addressed. Referring to the figure, if when body i is integrated to time t_c a possible collision with body 1 is detected, then the states of both bodies must be reset to the last point at which they were verified as separate. The work performed to integrate body 1 to its current state would be wasted in this case. Still, this scheme would likely result in a significant performance improvement on average.

8.3 Interpreted control and object encapsulation

One powerful application of simulation is the design and testing of control systems. Experience with *Impulse* has demonstrated that many potential problems with a control scheme can be detected and corrected through simulation alone. A disadvantage of *Impulse's* current architecture is that control laws must be specified in C, compiled, and linked with the rest of the simulator code before they can be tested. This delay in the design-test loop quickly becomes a burden. Even more serious is the portability problem created when different versions of *Impulse* have different compiled control laws. A better approach would be to specify high level control laws in an interpreted language, such as Scheme or Java [GM95]. At a control update event, the necessary dynamic states would be passed to the interpreter, which would execute the control law, and pass back control forces and torques. More general state information, such as that arising during planning, could be kept in static data structures on the interpreted side. For efficiency, the low level controllers, and springs and dampers, could still be implemented as compiled C routines called by the high level controller. The interpreter provides a completely general language for specifying control laws. One concern is the impact on execution time, although from Table 7.2 the outlook appears good. The table indicates that a very small fraction of the time is spent executing control laws. Of this fraction, most of the time is devoted to executing low level control functions, such as the PD update routine. For these examples, a reasonable estimate is that less than 1% of the time is spent executing user-defined control routines. If the interpreted versions of these routines ran ten times slower than the compiled versions, execution time would only increase 10%.

Once the control laws are being written in an interpreted language, it makes sense to combine the entire object specification in such a language. Rather than have the geometry of an object specified in some fixed format data file, it could be specified by a Java program that executed calls to compiled library routines, like `createVertex(x,y,z)` or `createEdge(v1,v2)`. Such calls could be collected in a list, or put inside loops so that an interpreted program could algorithmically build up the geometry. The line between program and data becomes blurred; there is simply a paragraph of code that generates the geometry somehow. Other important properties for dynamic simulation could also be specified by the same program, such as masses or densities of objects, coefficients of friction, and so on. Even the rendering model could be specified this way, although there may be more reason to keep that model in a fixed, standardized format for efficiency purposes. Leveraging off the language interpreter, this object specification format is completely general and portable. Users at different sites could each create full descriptions of creatures, including their geometries, physical properties, rendering models, and behaviors implemented through control systems. Scripts describing these creatures could be submitted to a common simulation engine; the creatures would inhabit a common world and be free to interact with each other. Graphical results could be sent back to the remote sites, where the creature designs could be evaluated and refined (Figure 8.2).

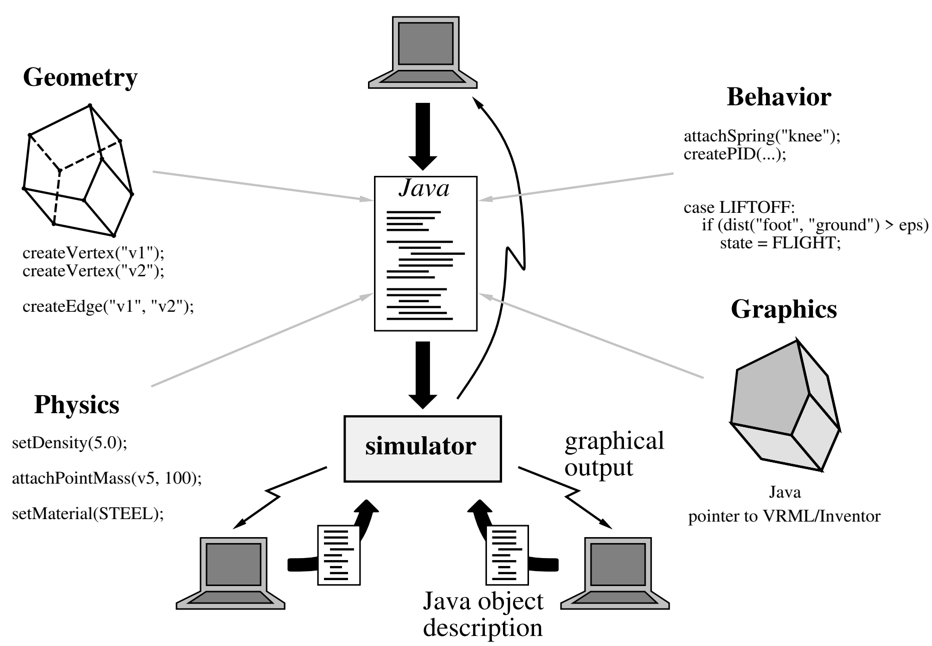

Figure 8.2: *Interpreted object specification allows generality and portability.*

Taking the logical next step, why make the designer program at all? Consider building a creature to inhabit a virtual world. A number of systems have been proposed that allow a user to interactively snap together kinematic structures subject to dynamic constraints [BB88, Gle94, IC87, WGW90]. One could envision such a system, augmented with widgets representing various types of low level controllers that could be attached to the creature. Various types of sensors, for example, compasses or joint encoders, could be attached as well. The sensors could be very simple, like encoders that return a joint position, or quite complex, like a vision sensor that returns a binary image from a camera mounted on the front of the creature. After the geometry and sensing and control systems were constructed, a high level control system could be specified, again using a graphical user interface. One paradigm would be interactive design of state machine control systems, where the actions to be taken at each state and the conditions for state transitions are built up from primitives based on the control and sensing systems that are on board (Figure 8.3). Finally, at the push of a button, a complete description of the creature, in the form of a Java

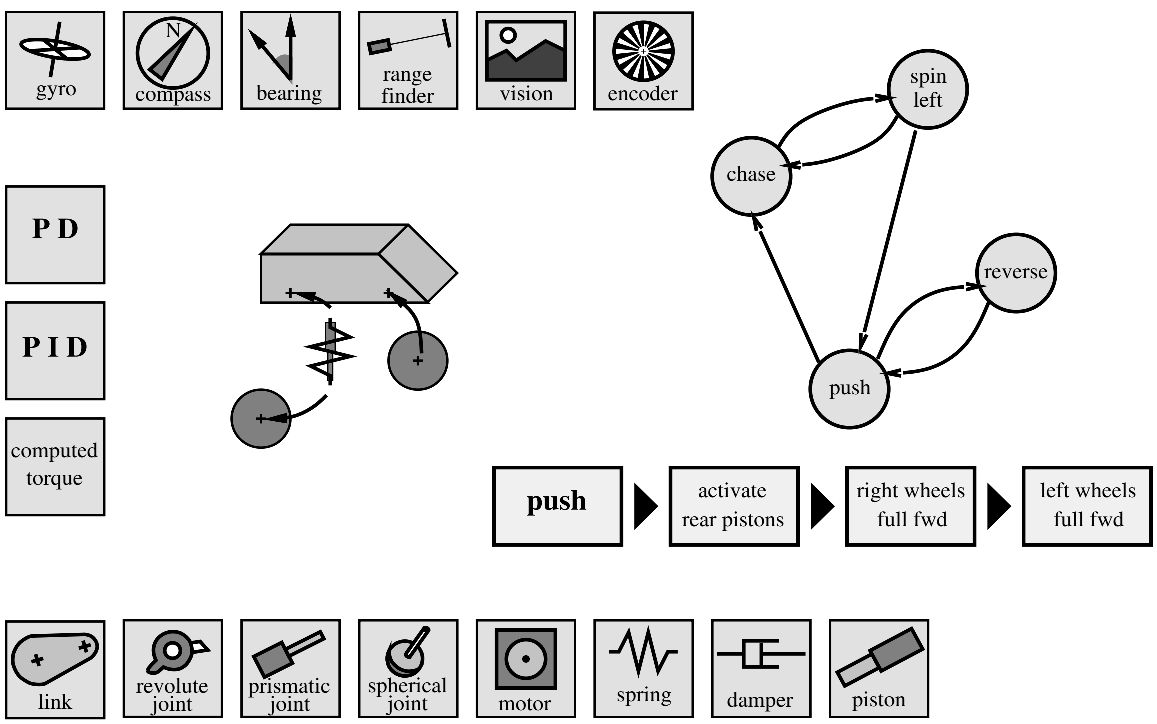

Figure 8.3: *A conception of a creature design environment. Through a graphical user interface, a designer completely describes the geometry, physics, control and sensing, rendering model, and behavior for a creature. A script describing the creature is automatically generated and ready for use in a physical simulation.*

program, is automatically generated, and ready for deployment into a virtual world. Many details need to be worked out, but such a system is entirely possible and would greatly accelerate the creation of interesting virtual worlds populated by intelligent creatures.

8.4 Physical simulation for animation and VR

In recent years, the computer graphics community has increasingly turned to physical models for generating realistic animations. Several systems have been developed which model a particular type of creature, often with intelligence and goals, interacting within its environment; just a few examples are in [Rey87, MZ90, RH91, Mil91, NM93, TT94, Sim94, HWBO95]. The animations these systems produce are extremely compelling, and one can imagine the richness of a virtual world populated by these sorts of autonomous creatures, operating under independent control systems and interacting in complex ways. This has yet to happen, however, largely because physically based animation systems have each been specialized for particular types of creatures and environments. Miller's system is for animating snakes, Tu's and Terzopoulos's system is for fish, Raibert's and Hodgins's system only models contact with a ground plane, and so on.

This is in no way a criticism of these systems, since traditionally such restrictions have been necessary to obtain reasonable speeds. This condition is changing however. An impulse-based simulator can efficiently and accurately simulate three-dimensional physical systems comprising rigid and articulated bodies: a fairly general class. Fast and robust *general* collision detection and physically accurate collision response facilitate realistic interactions among different components or creatures, no matter how they come together. The designer of a creature does not have to worry about the variety of contact interactions it might encounter in a virtual world; correct contact forces will be computed. The designer can then consider the behavior system at a higher level, as one would if designing the controller for a robot in the real world. Many questions and challenges remain, however, *Impulse* has shed some light on the possibilities that a general purpose dynamic simulator could create. Such a simulator is needed before the work of so many can be combined in a virtual environment inhabited by diverse, autonomous agents.

Appendix A

Mathematical Preliminaries

A.1 Vectors, matrices, and frames

In this text, a *frame* designates a right-handed, three-dimensional coordinate system. Frames are usually designated by scripted uppercase letters, such as $\mathcal{F}$. The right-handed requirement means that the canonical basis vectors satisfy

$$\mathbf{i} \times \mathbf{j} = \mathbf{k}.$$

The components of a vector are always expressed relative to the basis vectors of some such frame, which may be implied or stated explicitly.

It is often necessary to convert the coordinates of a vector expressed in one frame to coordinates expressed in another frame. Let $\mathbf{v}_{\mathcal{F}}$ and $\mathbf{v}_{\mathcal{G}}$ be 3×1 column vectors comprising the coordinates of vector $\mathbf{v}$ relative to frames $\mathcal{F}$ and $\mathcal{G}$, respectively. Let $\mathbf{r}_x, \mathbf{r}_y$, and $\mathbf{r}_z$ be the 3×1 column vector comprising the coordinates of frame $\mathcal{F}'s$ $\mathbf{i}$,$\mathbf{j}$, and $\mathbf{k}$ basis vectors relative to frame $\mathcal{G}$, respectively. Then,

$$\mathbf{v}_{\mathcal{G}} = \underbrace{\left[\begin{array}{c|c|c} \mathbf{r}_x & \mathbf{r}_y & \mathbf{r}_z \end{array}\right]}_{\mathbf{R}} \mathbf{v}_{\mathcal{F}}.$$

The 3×3 matrix $\mathbf{R}$ is called a *change-of-basis matrix.* Under the assumption that the two bases are orthonormal, right-handed bases, the change-of-basis is always a rotation matrix. This is an orthogonal matrix, meaning that its columns are all of unit length and mutually perpendicular. The same can be said of its rows. Furthermore, its determinant is +1, and it possesses the extremely useful property that

$$\mathbf{R}^{-1} = \mathbf{R}^T.$$

Matrices must also be transformed to operate on vectors specified in different bases. For example, Let $\boldsymbol{\omega}$ and $\mathbf{L}$ be the coordinates of the rigid body's angular velocity and angular momentum, relative to a frame attached to the body. These coordinate vectors are related to each other through the 3×3 mass matrix, $\mathbf{I}$:

$$\mathbf{L} = \mathbf{I}\boldsymbol{\omega}.$$

Suppose now that the body's angular velocity and angular momentum coordinate vectors are expressed relative to a different frame, fixed in space. Call these fixed frame vectors $\boldsymbol{\omega}_0$ and $\mathbf{L}_0$. These two vectors can still be related through a matrix $\mathbf{I}_0$, given by

$$\mathbf{I}_0 = \mathbf{R}\mathbf{I}\mathbf{R}^T,$$

where $\mathbf{R}$ is the rotation matrix taking coordinate vectors in the body frame to coordinate vectors in the fixed frame. The matrix $\mathbf{I}_0$ represents the same linear mapping as $\mathbf{I}$ does; it is just coordinatized relative to a different basis. Justification for the transformation above, can be seen from

$$\begin{aligned}
\mathbf{L}_0 &= \mathbf{I}_0\boldsymbol{\omega}_0 \\
&= \left(\mathbf{R}\mathbf{I}\mathbf{R}^T\right)(\mathbf{R}\boldsymbol{\omega}) \\
&= \mathbf{R}\mathbf{I}\boldsymbol{\omega} \\
&= \mathbf{R}\mathbf{L}.
\end{aligned}$$

Thus, the angular momentum vector computed in the fixed frame using $\mathbf{I}_0$ and $\boldsymbol{\omega}_0$ is the same as that computed in the body frame using $\mathbf{I}$ and $\boldsymbol{\omega}$, but with the coordinates expressed in the new frame.

Matrices can also represent the coordinates of second-order tensors that map two vectors into a scalar. These coordinate matrices are transformed to a new frame in the same way. See [Cul72] for details.

A.2 Representing cross products as matrices

In the vector space $\mathbb{R}^3$, the cross product ($\times$) is an operator taking two vectors to a third vector. In coordinates, the operation can be described by

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}.$$

One can equivalently write

$$\mathbf{a} \times \mathbf{b} = \tilde{\mathbf{a}}\mathbf{b},$$

where $\tilde{\mathbf{a}}$ is the 3×3 skew-symmetric matrix

$$\tilde{\mathbf{a}} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}.$$

In this way, $\tilde{\mathbf{a}}$ can be thought of as a transformation from $\mathbb{R}^3$ to itself, taking each vector $\mathbf{v}$ to the cross product $\mathbf{a} \times \mathbf{v}$, where $\mathbf{a}$ is a fixed vector. Representing the cross product operation as a matrix is often useful in studying equations arising in dynamics; this thesis uses this representation frequently, denoting cross product matrices with the standard tilde notation. The matrix $\tilde{\mathbf{a}}$ is skew-symmetric:

$$\tilde{\mathbf{a}}^T = -\tilde{\mathbf{a}}.$$

Let the matrix $\tilde{\mathbf{a}}_{\mathcal{F}}$ correspond to the "cross product by $\mathbf{a}$" operation, in frame $\mathcal{F}$. This same operation may be expressed in a new frame $\mathcal{G}$ using the matrix transformation rule described above. If $\mathbf{R}$ is the rotation matrix transforming coordinate vectors in $\mathcal{F}$ to coordinate vectors in $\mathcal{G}$, then the operation is described in the new frame by

$$\tilde{\mathbf{a}}_{\mathcal{G}} = \mathbf{R}\tilde{\mathbf{a}}_{\mathcal{F}}\mathbf{R}^T.$$

The matrix $\tilde{\mathbf{a}}_{\mathcal{G}}$ represents the same operation as $\tilde{\mathbf{a}}_{\mathcal{F}}$, except it operates on coordinate vectors resolved in frame $\mathcal{G}$ rather than frame $\mathcal{F}$. Since

$$\mathbf{a}_{\mathcal{G}} = \mathbf{R}\mathbf{a}_{\mathcal{F}},$$

the transformation rule for cross product matrices is often written as

$$\widetilde{(\mathbf{R}\mathbf{a})} = \mathbf{R}\tilde{\mathbf{a}}\mathbf{R}^T.$$

Cross product matrices provide a convenient way of writing the derivative of a time varying rotation matrix.

Theorem 24 (Derivative of a rotation matrix) *Let $\mathcal{B}$ be a frame rotating relative to an inertial frame $\mathcal{O}$, with instantaneous angular velocity $\boldsymbol{\omega}(t)$. Let $\mathbf{R}(t)$ be the rotation matrix that transforms vector coordinates in $\mathcal{B}$ to vector coordinates in $\mathcal{O}$. If $\boldsymbol{\omega}_o(t)$ is the representation of $\boldsymbol{\omega}(t)$ in frame $\mathcal{O}$ coordinates, and $\boldsymbol{\omega}_b(t)$ is the representation of $\boldsymbol{\omega}(t)$ in frame $\mathcal{B}$ coordinates, then*

$$\begin{aligned}\dot{\mathbf{R}} &= \tilde{\boldsymbol{\omega}}_o\mathbf{R},\\ \dot{\mathbf{R}} &= \mathbf{R}\tilde{\boldsymbol{\omega}}_b.\end{aligned}$$

Proof: Let $\mathbf{r}$ be an arbitrary vector that is fixed in, and therefore rotates with, frame $\mathcal{B}$. Let $\mathbf{r}_o$ and $\mathbf{r}_b$ be the coordinate representations of $\mathbf{r}$ in $\mathcal{O}$ and $\mathcal{B}$. One can compute the derivative $\dot{\mathbf{r}}_o$ in several ways:

$$\begin{aligned}\dot{\mathbf{r}}_o &= \frac{d}{dt}(\mathbf{R}\mathbf{r}_b) = \dot{\mathbf{R}}\mathbf{r}_b + \mathbf{R}\dot{\mathbf{r}}_b = \dot{\mathbf{R}}\mathbf{r}_b\\ \dot{\mathbf{r}}_o &= \boldsymbol{\omega}_o \times \mathbf{r}_o = \boldsymbol{\omega}_o \times (\mathbf{R}\mathbf{r}_b) = \tilde{\boldsymbol{\omega}}_o\mathbf{R}\mathbf{r}_b\\ \dot{\mathbf{r}}_o &= \mathbf{R}(\boldsymbol{\omega}_b \times \mathbf{r}_b) = \mathbf{R}\tilde{\boldsymbol{\omega}}_b\mathbf{r}_b\end{aligned}$$

The first equation follows from straight differentiation; the second and third follow from the formula for the velocity of a point on a rotating body, performing the cross product in frames $\mathcal{O}$ and $\mathcal{B}$, respectively. Equating the right hand sides of these three equations,

$$\dot{\mathbf{R}}\mathbf{r}_b = \tilde{\boldsymbol{\omega}}_o\mathbf{R}\mathbf{r}_b = \mathbf{R}\tilde{\boldsymbol{\omega}}_b\mathbf{r}_b.$$

Since $\mathbf{r}_b$ is arbitrary, the result follows. $\square$

A.3 Rigid body dynamics

All of the external influences on a rigid body may be summarized as a single force $\mathbf{f}(t)$ acting at the center of mass of the body, and a single moment $\boldsymbol{\tau}(t)$ applied to the body.

Let $\mathbf{r}(t)$ be the position of the center of mass of the body, and m the mass of the body. By Newton's second law,

$$\mathbf{f}(t) = \frac{d}{dt}\left[m\dot{\mathbf{r}}(t)\right] = m\ddot{\mathbf{r}}(t). \tag{A.1}$$

This relation can be rephrased as a first order ordinary differential equation by letting $\mathbf{v}(t)$ denote the velocity of the center of mass of the body. Then,

$$\begin{aligned} \dot{\mathbf{r}}(t) &= \mathbf{v}(t) \\ \dot{\mathbf{v}}(t) &= \frac{1}{m}\mathbf{f}(t). \end{aligned}$$

For any $\mathbf{f}(t)$, this first order system can always be numerically integrated to determine the motion of the center of mass over time. In many cases, particularly those described in this thesis, $\mathbf{r}(t)$ can be solved in closed form. Consider the case of a *ballistic* body, where gravity is the only external influence. The gravitational force acts downward through the center of mass, with constant magnitude mg, where g is the gravitational constant. Equation (A.1) becomes

$$-mg\mathbf{k} = m\ddot{\mathbf{r}}(t),$$

which upon integrating yields:

$$\mathbf{r}(t) = \begin{bmatrix} r_x(0) + v_x(0)t \\ r_y(0) + v_y(0)t \\ r_z(0) + v_z(0)t - \frac{1}{2}gt^2 \end{bmatrix}.$$

The vectors $\mathbf{r}(0)$ and $\mathbf{v}(0)$ are initial data needed to completely solve the ODE.

The other component of rigid body motion is rotation. Let $\boldsymbol{\omega}_o(t)$ be the angular velocity of the body, and $\boldsymbol{\tau}_0$ the total moment applied to the body. The rotational analog of (A.1) is

$$\boldsymbol{\tau}_o(t) = \frac{d}{dt}\left[\mathbf{I}_o(t)\boldsymbol{\omega}_o(t)\right]. \tag{A.2}$$

$\mathbf{I}_o(t)$ is often called the *mass matrix* or *inertia tensor*. It describes how mass is distributed in the body. The equation above is only valid when the vectors and matrices are expressed relative to a frame that it not rotating, like a frame $\mathcal{O}$ with origin at the center of mass of the body, but with axes that point in fixed directions. The o subscripts denote quantities expressed in this frame. It can be shown that

$$\mathbf{I}_o = \mathbf{R}(t)\mathbf{I}\mathbf{R}^T(t),$$

where $\mathbf{I}$ is the *body frame mass matrix*. This matrix describes how mass is distributed in the body, relative to a frame $\mathcal{B}$ with origin at the center of mass, and that rotates with the body. $\mathbf{I}$ is a constant diagonal matrix, and $\mathbf{R}(t)$ is the rotation matrix that maps vectors in $\mathcal{B}$ to vectors in $\mathcal{O}$. The above equation is just an example of a matrix change-of-basis transformation. If $\boldsymbol{\omega}(t)$ and $\boldsymbol{\tau}(t)$ are the angular velocity and applied moment vectors expressed in frame $\mathcal{B}$, (A.2) can be rewritten:

$$\begin{aligned}\mathbf{R}\boldsymbol{\tau}(t) &= \frac{d}{dt}\left[\left(\mathbf{R}(t)\mathbf{I}\mathbf{R}^T(t)\right)(\mathbf{R}(t)\boldsymbol{\omega}(t))\right] \\ &= \dot{\mathbf{R}}(t)\mathbf{I}\boldsymbol{\omega}(t) + \mathbf{R}(t)\mathbf{I}\dot{\boldsymbol{\omega}}(t) \\ &= \mathbf{R}(t)\tilde{\boldsymbol{\omega}}(t)\mathbf{I}\boldsymbol{\omega}(t) + \mathbf{R}(t)\mathbf{I}\dot{\boldsymbol{\omega}}(t).\end{aligned}$$

Multiplying both sides by $\mathbf{R}^T(t)$ and solving for $\dot{\boldsymbol{\omega}}(t)$,

$$\dot{\boldsymbol{\omega}}(t) = \mathbf{I}^{-1}\left[-\tilde{\boldsymbol{\omega}}(t)\mathbf{I}\boldsymbol{\omega}(t) + \boldsymbol{\tau}(t)\right]. \tag{A.3}$$

Writing the diagonal elements of $\mathbf{I}$ as I_x, I_y, and I_z,

$$\begin{bmatrix}\dot{\omega}_x(t)\\ \dot{\omega}_y(t)\\ \dot{\omega}_z(t)\end{bmatrix} = \begin{bmatrix}\frac{I_y-I_z}{I_x}\omega_y(t)\omega_z(t)\\ \frac{I_z-I_x}{I_y}\omega_z(t)\omega_x(t)\\ \frac{I_x-I_y}{I_z}\omega_x(t)\omega_y(t)\end{bmatrix} + \begin{bmatrix}\frac{\tau_x(t)}{I_x}\\ \frac{\tau_y(t)}{I_y}\\ \frac{\tau_z(t)}{I_z}\end{bmatrix}.$$

These are the Euler equations; they describe how body angular velocity evolves over time.

Often the equations of translation and rotation are grouped together and written:

$$\sum \mathbf{f}(t) = m\mathbf{a}(t) \tag{A.4}$$

$$\sum \boldsymbol{\tau}(t) = \mathbf{I}\boldsymbol{\alpha}(t) + \boldsymbol{\omega}(t) \times \mathbf{I}\boldsymbol{\omega}(t) \tag{A.5}$$

Equation (A.4) is just (A.1), with $\mathbf{a}(t)$ denoting linear acceleration, the second time derivative of position. Equation (A.5) is a restatement of (A.3), with $\boldsymbol{\alpha}(t)$ denoting angular acceleration, the time derivative of angular velocity. The summation signs are reminders that the *total* force and moment acting on the body should be used. Together, these are called the *Newton-Euler* equations; they completely characterize the dynamics of a rigid body. More detailed treatments of these topics are in [Gre88, MK86].

A.4 Quaternions and integration of orientation

For dynamic simulation, unit quaternions are usually the best representation for the orientation of a rigid body. Unlike Euler angles and related parameterizations, quaternions provide a parameterization with no singularities in the mapping to $SO(3)$. Unlike rotation matrices, quaternions are a minimum dimension nonsingular parameterization for $SO(3)$, and when the configuration point drifts off the manifold, it can easily be projected back onto it by normalizing the quaternion. This section derives the ODE that needs to be integrated for the orientation of a rigid body as a function of time. A more detailed presentation of quaternions is in [FP88].

Every element of $SO(3)$ can be identified with a rotation about some unit vector $\mathbf{u}$ by an angle θ. The matrix corresponding to this rotation is

$$\mathbf{R} = \begin{bmatrix} u_x^2 \text{vers } \theta + \cos\theta & u_y u_x \text{vers } \theta - u_z \sin\theta & u_z u_x \text{vers } \theta + k_y \sin\theta \\ u_x u_y \text{vers } \theta + k_z \sin\theta & u_y^2 \text{vers } \theta + \cos\theta & u_z u_y \text{vers } \theta - u_x \sin\theta \\ u_z u_z \text{vers } \theta - u_y \sin\theta & u_y u_z \text{vers } \theta + k_x \sin\theta & u_z^2 \text{vers } \theta + \cos\theta \end{bmatrix}, \tag{A.6}$$

where $\text{vers } \theta = 1 - \cos\theta$. This orientation can also be represented by the unit quaternion

$$\mathbf{q} = (q_s; q_x, q_y, q_z) = (\cos\frac{\theta}{2}; u_x \sin\frac{\theta}{2}, u_y \sin\frac{\theta}{2}, u_z \sin\frac{\theta}{2}).$$

This relation can be inverted to express $\mathbf{u}$ and trigonometric functions of θ in terms of the quaternion components:

$$\begin{aligned} \mathbf{u} &= \frac{1}{\sqrt{1-q_s^2}} \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \\ \cos\theta &= 2q_s^2 - 1 \\ \sin\theta &= 2q_s\sqrt{1-q_s^2} \\ \text{vers } \theta &= 2(1-q_s^2). \end{aligned}$$

Substituting these equivalences into (A.6) gives $\mathbf{R}$ in terms of quaternions:

$$\mathbf{R} = 2 \begin{bmatrix} q_s^2 + q_x^2 - \frac{1}{2} & q_y q_x - q_z q_s & q_z q_x + q_y q_s \\ q_x q_y + q_z q_s & q_s^2 + q_y^2 - \frac{1}{2} & q_z q_y - q_x q_s \\ q_x q_z - q_y q_s & q_y q_z + q_x q_s & q_s^2 + q_z^2 - \frac{1}{2} \end{bmatrix}. \tag{A.7}$$

If the quaternion components vary with time,

$$\dot{\mathbf{R}} = 2 \begin{bmatrix} 2(q_s\dot{q}_s + q_x\dot{q}_x) & \dot{q}_y q_x + q_y\dot{q}_x - \dot{q}_z q_s - q_z\dot{q}_s & \dot{q}_z q_x + q_z\dot{q}_x + \dot{q}_y q_s + q_y\dot{q}_s \\ \dot{q}_x q_y + q_x\dot{q}_y + \dot{q}_z q_s + q_z\dot{q}_s & 2(q_s\dot{q}_s + q_y\dot{q}_y) & \dot{q}_z q_y + q_z\dot{q}_y - \dot{q}_x q_s - q_x\dot{q}_s \\ \dot{q}_x q_z + q_x\dot{q}_z - \dot{q}_y q_s - q_y\dot{q}_s & \dot{q}_y q_z + q_y\dot{q}_z + \dot{q}_x q_s + q_x\dot{q}_s & 2(q_s\dot{q}_s + q_z\dot{q}_z) \end{bmatrix} \tag{A.8}$$

Suppose $\mathbf{R}(t)$ is the rotation matrix mapping vectors in the body frame to vectors in a fixed frame. If $\boldsymbol{\omega}$ is the angular velocity expressed in the body frame, then from Theorem 24,

$$\tilde{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix} = \mathbf{R}^T\dot{\mathbf{R}}.$$

Substituting (A.7) and (A.8) into the above equation produces a complex expression, but it simplifies greatly after applying the unit quaternion restriction and its derivative:

$$\begin{aligned} q_s^2 + q_x^2 + q_y^2 + q_z^2 &= 1 \\ q_s\dot{q}_s + q_x\dot{q}_x + q_y\dot{q}_y + q_z\dot{q}_z &= 0. \end{aligned}$$

With these substitutions, the components of $\boldsymbol{\omega}$ are easily deduced:

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ 0 \end{bmatrix} = 2 \begin{bmatrix} -q_x & +q_s & +q_z & -q_y \\ -q_y & -q_z & +q_s & +q_x \\ -q_z & +q_y & -q_x & +q_s \\ +q_s & +q_x & +q_y & +q_z \end{bmatrix} \begin{bmatrix} \dot{q}_s \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{bmatrix}.$$

The last row of the above equation is just the derivative of the unit quaternion constraint; it is added to make the matrix square. This matrix is always invertible (the determinant is always -1). Inverting the equation gives

$$\begin{bmatrix} \dot{q}_s \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -x & -y & -z & +s \\ +s & -z & +y & +x \\ +z & +s & -x & +y \\ -y & +x & +s & +z \end{bmatrix} \begin{bmatrix} w_x \\ w_y \\ w_z \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -x & -y & -z \\ +s & -z & +y \\ +z & +s & -x \\ -y & +x & +s \end{bmatrix} \boldsymbol{\omega}.$$

This final equation expresses the derivatives of the quaternion components in terms of the angular velocities. It is used by the integrator to evolve the orientation, which is parameterized by a quaternion.

Bibliography

[ACH+94] O. Ahmad, J. Cremer, S. Hansen, J. Kearney, and P. Willemson. Hierarchical, concurrent state machines for behavior modeling and scenario control. In *Proceedings of 1994 Conference on AI, Simulation, and Planning in High Autonomy Systems*, December 1994.

[Bar89] David Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics*, 23(3):223–232, July 1989.

[Bar90] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics*, 24(4):19–28, August 1990.

[Bar91] David Baraff. Coping with friction for non-penetrating rigid body simulation. *Computer Graphics*, 25(4):31–40, August 1991.

[Bar92] David Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. PhD thesis, Department of Computer Science, Cornell University, March 1992.

[Bar94] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *SIGGRAPH Conference Proceedings*. ACM Press, 1994.

[Bar96] David Baraff. Linear-time dynamics using lagrange multipliers. In *SIGGRAPH Conference Proceedings*. ACM Press, 1996.

[BB88] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22(4):179–188, August 1988.

[BC96] Dina R. Berkowitz and John Canny. Designing parts feeders using dynamic simulation. In *International Conference on Robotics and Automation*, pages 1127–1132. IEEE, April 1996.

[BD86] William E. Boyce and Richard C. DiPrima. *Elementary Differential Equations and Boundary Value Problems.* John Wiley & Sons, Inc., New York, fourth edition, 1986.

[BDMM94] K. Böhringer, B. Donald, R. Mihailovich, and N. MacDonald. Sensorless manipulation using massively parallel microfabricated actuator arrays. In *International Conference on Robotics and Automation.* IEEE, May 1994.

[BJO86] H. Brandl, R. Johanni, and M. Otter. A very efficient algorithm for the simulation of robots and similar multibody systems. In *Proceedings of IFAC/IFIP/IMACS International Symposium on the Theory of Robots*, December 1986.

[BK94] Vivek Bhatt and Jeff Koechling. Classifying dynamic behavior during three dimensional frictional rigid body impact. In *International Conference on Robotics and Automation.* IEEE, May 1994.

[BK96a] Vivek Bhatt and Jeff Koechling. Partitioning the parameter space according to different behavior during 3d impacts. *Transactions of ASME : Journal of Applied Mechanics*, 1996. (To appear).

[BK96b] Vivek Bhatt and Jeff Koechling. Three dimensional frictional rigid body impact. *Transactions of ASME : Journal of Applied Mechanics*, 1996. (To appear).

[BPM82] Geoffrey Boothroyd, Corrado Poli, and Laurence E. Murch. *Automatic Assembly.* Marcel Dekker, Inc., 1982.

[Bra91] Raymond M. Brach. *Mechanical Impact Dynamics; Rigid Body Collisions.* John Wiley & Sons, Inc., 1991.

[Bro86] Rodney Brooks. A robust layered intelligent control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[BS95] Richard W. Bukowski and Carlo H. Séquin. Object associations: A simple and practical approach to virtual 3d manipulation. In *Symposium on Interactive 3D Graphics*, pages 131–138, New York, 1995. ACM Press.

[Can84] John Canny. Collision detection for moving polyhedra. Technical Report MIT A.I. Lab Memo 806, Massachusetts Institute of Technology, October 1984.

[CEC96] Alan D. Christiansen, Andrea Dunham Edwards, and Carlos A. Coello Coello. Automated design of part feeders using a genetic algorithm. In *International Conference on Robotics and Automation*. IEEE, 1996.

[CLMP95] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav K. Ponamgi. I-collide: An interactive and exact collision detection system for large-scaled environments. In *Symposium on Interactive 3D Graphics*, pages 189–196. ACM Siggraph, ACM Siggraph, April 1995.

[CPS92] Richard K. Cottle, Jong-Shi Pang, and Richard E. Stone. *The linear complementarity problem*. Academic Press, 1992.

[Cra89] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison Wesley Series in Electrical and Computer Engineering. Addison-Wesley, Reading, MA, second edition, 1989.

[CS89] James F. Cremer and A. James Stewart. The architecture of newton, a general-purpose dynamics simulator. In *International Conference on Robotics and Automation*, pages 1806–1811. IEEE, May 1989.

[Cul72] Charles G. Cullen. *Matrices and Linear Transformations*. Dover Publications, Inc., New York, second edition, 1972.

[Dev82] Jay L. Devore. *Probability & Statistics for Engineering and the Sciences*. Brooks/Cole Publishing Company, Monterey, California, 1982.

[dJB94] Javier Garcìa de Jalòn and Eduardo Bayo. *Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time Challenge*. Springer-Verlag, 1994.

[Ede83] H. Edelsbrunner. A new approach to rectangle intersections, part i. *International Journal of Computational Mathematics*, 13:209–219, 1983.

[EP86] C. H. Edwards, Jr. and David E. Penney. *Calculus and Analytic Geometry*. Prentice-Hall, Inc., Englewood Cliffs, second edition, 1986.

[Far90] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, Inc., San Diego, second edition, 1990.

[Fea83] R. Featherstone. The calculation of robot dynamics using articulated-body inertias. *International Journal of Robotics Research*, 2(1):13–30, 1983.

[FP88] Janez Funda and Richard P. Paul. A comparison of transforms and quaternions in robotics. In *International Conference on Robotics and Automation*, pages 886–891. IEEE, May 1988.

[FR88] R.T. Farouki and V.T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(1):1–26, June 1988.

[GC95] Ken Goldberg and John Craig. Estimating throughput for a flexible part feeder: Simulation and experiments. In *International Symposium of Experimental Robotics*. Stanford University, June 1995.

[Get94] N. H. Getz. Control of balance for a nonlinear nonholonomic nonminimum-phase model of a bicycle. In *American Control Conference*, Baltimore, June 1994. American Automatic Control Council.

[GJK88] Elmer G. Gilbert, Daniel W. Johnson, and S. Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, April 1988.

[Gle94] Michael L. Gleicher. *A Differential Approach to Graphical Interaction*. PhD thesis, School of Computer Science, Carnegie Mellon University, November 1994.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *SIGGRAPH Conference Proceedings*. ACM Press, 1996.

[GM95] James Gosling and Henry McGilton. The java language environment: A white paper. Technical report, Sun Microsystem, 1995.

[Gre88] Donald T. Greenwood. *Principles of Dynamics*. Prentice-Hall, Inc., Englewood Cliffs, second edition, 1988.

[Hah88] James K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, August 1988.

[HBZ90] Brian Von Herzen, Alan H. Barr, and Harold R. Zatz. Geometric collisions for time-dependent parametric surfaces. In *SIGGRAPH Conference Proceedings*, pages 39–48. ACM Press, 1990.

[HJ91] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, 1991.

[Hub94] Philip M. Hubbard. *Collision Detection for Interactive Graphics Applications*. PhD thesis, Department of Computer Science, Brown University, October 1994.

[Hub96] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3), July 1996.

[HWBO95] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. *Computer Graphics*, pages 71–78, 1995. Siggraph Conference Proceedings.

[IC87] Paul M. Isaacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. In *SIGGRAPH Conference Proceedings*, pages 215–224. ACM Press, 1987.

[JR] A. Jain and G. Rodriguez. Lecture notes on the spatial operator algebra for multibody dynamics.

[Kel86] J. B. Keller. Impact with friction. *Journal of Applied Mechanics*, 53, March 1986.

[Kha87] Oussama Khatib. A unified approach for the motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, RA-3(1):43–53, February 1987.

[LC91] Ming C. Lin and John F. Canny. A fast algorithm for incremental distance calculation. In *International Conference on Robotics and Automation*, pages 1008–1014. IEEE, May 1991.

[Lil93] Kathryn W. Lilly. *Efficient Dynamic Simulation of Robotic Mechanisms*. Kluwer Academic Publishers, Norwell, 1993.

[Lin93] Ming C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley, December 1993.

[LK84] Sheue-ling Lien and James T. Kajiya. A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *IEEE Computer Graphics and Applications*, 4(10):35–41, October 1984.

[LM93] M.C. Lin and Dinesh Manocha. Interference detection between curved objects for computer animation. In *Models and Techniques in Computer Animation*, pages 431–57. Springer-Verlag, 1993.

[LM95] Andrew D. Lewis and Richard M. Murray. Variational principles in constrained systems: Theory and experiments. *International Journal of Nonlinear Mechanics*, 1995.

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

[Löt81] Per Lötstedt. Coulomb friction in two-dimensional rigid body systems. *Zeitschrift fur Angewandte Mathematik und Mechanik*, 61(12):605–15, 1981.

[Löt84] Per Lötstedt. Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. *SIAM Journal of Scientific Statistical Computing*, 5(2):370–93, June 1984.

[LR82a] Yong Tsui Lee and Aristides A. G. Requicha. Algorithms for computing the volume and other integral properties of solids. I. Known methods and open issues. *Communications of the ACM*, 25(9):635–41, September 1982.

[LR82b] Yong Tsui Lee and Aristides A. G. Requicha. Algorithms for computing the volume and other integral properties of solids. II. A family of algorithms based on representation conversion and cellular approximation. *Communications of the ACM*, 25(9):642–50, September 1982.

[LRK94] Paul U. Lee, Diego C. Ruspini, and Oussama Khatib. Dynamic simulation of interactive robotic environment. In *International Conference on Robotics and Automation*, pages 1147–1152. IEEE, May 1994.

[MC95a] Brian Mirtich and John Canny. Impulse-based dynamic simulation. In K. Goldberg, D. Halperin, J.C. Latombe, and R. Wilson, editors, *The Algorithmic Foundations of Robotics*. A. K. Peters, Boston, MA, 1995. Proceedings from the workshop held in February, 1994.

[MC95b] Brian Mirtich and John Canny. Impulse-based simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*, pages 181–188, New York, 1995. ACM Press.

[Mil91] Gavin Miller. Goal-directed animation of tubular articulated figures or how snakes play golf. In Norman A. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 233–241, San Mateo, CA, 1991. Morgan Kaufmann Publishers, Inc. From the workshop held April 6–7, 1989.

[Mir96] Brian Mirtich. Fast and accurate computation of polyhedral mass properties. *Journal of Graphics Tools*, 1(1), 1996.

[MK86] J.L. Meriam and L.G. Kraige. *Engineering Mechanics Volume 2: Dynamics*. John Wiley & Sons, Inc., New York, 1986.

[MR94] J. Marsden and T. Ratiu. *Introduction to Mechanics and Symmetry: a basic exposition of classical mechanical systems*. Springer-Verlag, New York, 1994.

[MW88] Matthew Moore and Jane Wilhems. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, August 1988.

[MZ90] Michael McKenna and David Zeltzer. Dynamic simulation of autonomous legged locomotion. *Computer Graphics*, pages 29–38, 1990. Siggraph Conference Proceedings.

[MZG+96] Brian Mirtich, Yan Zhuang, Ken Goldberg, John Craig, Rob Zanutta, Brian Carlisle, and John Canny. Estimating pose statistics for robotic part feeders. In

International Conference on Robotics and Automation, pages 1140–1146. IEEE, April 1996.

[NM93] J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. *Computer Graphics*, pages 343–350, 1993. Siggraph Conference Proceedings.

[NW78] James L. Nevins and Daniel E. Whitney. Computer-controlled assembly. *Scientific American*, 1978.

[Ove92] M. Overmars. Point location in fat subdivisions. *Information Processing Letters*, 44:261–265, 1992.

[Pau95] Eric Paulos. Parallel impulse based dynamic simulation. Available from WWW site `http://robotics.eecs.berkeley.edu/ paulos/IMP/`, May 1995.

[PML95] Madhav K. Ponamgi, Dinesh Manocha, and Ming C. Lin. Incremental algorithms for collision detection between solid models. In *Proceedings of Third ACM Symposium on Solid Modeling and Applications*, pages 293–304, New York, May 1995. ACM Press.

[PS88] Michael A. Peshkin and Arthur C. Sanderson. The motion of a pushed, sliding workpiece. *IEEE Journal of Robotics and Automation*, 4(6):569–598, December 1988.

[PS89] Michael A. Peshkin and Arthur C. Sanderson. Minimization of energy in quasi-static manipulation. *IEEE Transactions on Robotics and Automation*, 5(1):53–60, February 1989.

[PT96] J.S. Pang and J.C. Trinkle. Complementarity formulations and existence of solutions of dynamic multi-rigid-body contact problems with coulomb friction. *Mathematical Programming*, 1996. (To appear).

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian R. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, second edition, 1992.

[Rai86] Marc H. Raibert. *Legged Robots that Balance.* The MIT Press series in artificial intelligence. MIT Press, Cambridge, 1986.

[RB] Dan S. Reznik and Stan W. Brown. Dynamic simulation of micro-actuated motion arrays with application to design optimization. Submitted to *1997 IEEE International Conference on Robotics and Automation*.

[Rey87] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, pages 25–34, 1987. Siggraph Conference Proceedings.

[RG93] Anil Rao and Ken Goldberg. Planning grasps for a pivoting gripper. Technical report, USC Institute for Robotics and Intelligent Machines, October 1993. USC Techreport IRIS-93-313.

[RH91] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, pages 349–358, 1991. Siggraph Conference Proceedings.

[RK] Diego C. Ruspini and Oussama Khatib. Simpact. Available from WWW site `http://flamingo.stanford.edu/users/ruspini/simpact.html`.

[RKG95] A. Rao, D. Kriegman, and K. Goldberg. Complete algorithms for reorienting polyhedral parts using a pivoting gripper. In *International Conference on Robotics and Automation*. IEEE, May 1995.

[Rou05] Edward J. Routh. *Elementary Rigid Dynamics*. Macmillan, London, 1905.

[Sim94] Karl Sims. Evolving virtual creatures. *Computer Graphics*, pages 15–22, 1994. Siggraph Conference Proceedings.

[SSCK94] K. Sridharan, H.E. Stephanou, K.C. Craig, and S.S. Keerthi. Distance measures on intersecting objects and their applications. *Information Processing Letters*, 51(4):181–8, August 1994.

[Ste82] Sherman K. Stein. *Calculus and Analytic Geometry*. McGraw-Hill, Inc., New York, third edition, 1982.

[Str91] W. J. Stronge. Unraveling paradoxical theories for rigid body collisions. *Journal of Applied Mechanics*, 58:1049–1055, December 1991.

[SWF+93] John M. Snyder, Adam R. Woodbury, Kurt Fleischer, Bena Currin, and Alan H. Barr. Interval methods for multi-point collisions between time-dependent curved surfaces. In *SIGGRAPH Conference Proceedings*, pages 321–333. ACM Press, 1993.

[TPSL96] J.C. Trinkle, J.S. Pang, S. Sudarsky, and G. Lo. On dynamic multi-rigid-body contact problems with coulomb friction. *Zeitschrift fur Angewandte Mathematik und Mechanik*, 1996. (To appear).

[TT94] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. *Computer Graphics*, pages 43–50, 1994. Siggraph Conference Proceedings.

[Van91] George Van**ec**ek, Jr. Brep-index: A multi-dimensional spatial partitioning tree. *International Journal of Computational Geometry and Applications*, 1(3):243–262, September 1991.

[WB82] C. Ray Wylie and Louis C. Barrett. *Advanced Engineering Mathematics*. McGraw-Hill, Inc., New York, fifth edition, 1982.

[WGPB96] Jeff Wiegley, Ken Goldberg, Mike Peshkin, and Mike Brokowski. A complete algorithm for designing passive fences to orient parts. In *International Conference on Robotics and Automation*, pages 1133–1139. IEEE, April 1996.

[WGW90] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. *Computer Graphics*, 24(2):11–22, March 1990.

[WM87] Yu Wang and Matthew T. Mason. Modeling impact dynamics for robotic operations. In *International Conference on Robotics and Automation*, pages 678–685. IEEE, May 1987.

[WO82] M.W. Walker and D.E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, 104:205–211, September 1982.

[WRG92] Jeff Wiegley, Anil Rao, and Ken Goldberg. Computing a statistical distribution of stable poses for a polyhedron. In *30th Annual Allerton Conference on Communications, Control, and Computing*, 1992.

[Zhu96] Yan Zhuang. Simulation of linear electrostatic stepper motor (a mems device) using impulse. Available from WWW site `http://www.cs.berkeley.edu/ yzhuang/projects/cs287/report.html`, May 1996.