Dynamic Simulation of Non-Penetrating Rigid Bodies

David Baraff
Ph.D Thesis

92-1275
March 1992

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

DYNAMIC SIMULATION OF NON-PENETRATING RIGID BODIES

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
David Baraff

May 1992

© David Baraff 1992
ALL RIGHTS RESERVED

DYNAMIC SIMULATION OF NON-PENETRATING RIGID BODIES

David Baraff, Ph.D.
Cornell University 1992

This thesis examines the problems and difficulties in the forward dynamic simulation of rigid bodies subject to non-penetration constraints. By adopting a simple but well-defined model of rigid body dynamics, we are able to focus on and gain insight into some of the inherent difficulties of rigid body simulation. Additionally, computationally practical solutions to some of the problems encountered in this thesis are presented.

Enforcing non-penetration constraints is essentially a two step process. The first step of the simulation involves the detection of potential contacts between bodies. This thesis presents collision detection algorithms for the dynamic simulation of bodies that are composed of both polyhedra and convex closed curved surfaces. The collision detection algorithms exploit temporal coherence to achieve fast running times and are a practical solution to the problem of collision detection during simulation.

The second step of the simulation involves the computation of the contact forces between bodies that maintain the non-penetration constraint. This thesis considers first the problem of computing contact forces between a pair of bodies that contact at a point without friction. A mathematical formulation for the contact force between the bodies is presented, and then modified to yield a formulation that is

computationally practical for use in a simulator. After considering the dynamics of single point contacts, systems with multiple contacts are considered both in terms of computational complexity measures and practical solution methods. The methods used in this thesis to compute constraint forces are also theoretically and practically compared with a popular method for preventing inter-penetration called the "penalty method".

After considering frictionless systems, this thesis considers systems of bodies that behave according to the classical Coulomb model of friction (which includes both sliding and dry friction). This leads us to consider systems in which there are no solutions to the classical constraint force equations, as well as systems which admit multiple solutions for the constraint force equations and whose subsequent behavior is thus indeterminate. Both computational and practical complexity results for simulating such systems are discussed.

Table of Contents

1 Introduction **1**
- 1.1 Thesis concerns and restrictions . . . 2
- 1.2 Thesis overview . . . 4

2 A Simple Dynamics Problem with Non-penetration Constraints **5**
- 2.1 Unconstrained motion . . . 5
- 2.2 Equality constrained motion . . . 6
- 2.3 Inequality constrained motion . . . 7
- 2.4 The penalty method . . . 8
- 2.5 Exact solution methods . . . 11
  - 2.5.1 The equality constrained problem . . . 11
  - 2.5.2 The inequality constrained problem . . . 15

3 Simulation Overview **21**
- 3.1 Updating the current state . . . 22
- 3.2 Collision detection . . . 24
- 3.3 Contact point determination . . . 26
- 3.4 Impulse computation . . . 27
- 3.5 Constraint/friction force determination . . . 28

4 Summary of Previous Work **30**
- 4.1 Collision/contact determination . . . 30
- 4.2 Impulse computation . . . 32
- 4.3 Penalty methods for constraint forces . . . 34
- 4.4 Exact methods for constraint forces . . . 35
- 4.5 Friction . . . 36

5 Collision/Contact Determination **38**
- 5.1 Convex polyhedra . . . 40
- 5.2 Convex curved surfaces . . . 44

5.2.1 Extremal points of curved surfaces . . . 45
5.2.2 Polyhedron/curved surface determination . . . 50
5.3 Bounding boxes . . . 52
5.3.1 The one-dimensional case . . . 52
5.3.2 The three-dimensional case . . . 55

6 Local Motion Constraints **57**
6.1 Configuration Space . . . 58
6.2 Projections onto S . . . 60
6.3 Calculating $\partial^2 C/\partial \overline{p}^2$ . . . 63
6.4 Computing $\ddot{C}(\overline{x}(t_0))$ directly . . . 68
6.5 Predicting collision times . . . 73
6.6 Rolling contact . . . 74
6.7 Ill-conditioned Jacobians . . . 75

7 The Penalty Method **83**
7.1 Equality constrained motion . . . 84
7.1.1 The physically correct motion . . . 85
7.1.2 Motion due to the penalty method . . . 87
7.2 Inequality constrained motion . . . 92

8 Frictionless Systems **94**
8.1 Existence and uniqueness of solutions . . . 97
8.2 Implementation issues . . . 98

9 The Coulomb Friction Model **101**
9.1 Coordinate geometry at a contact point . . . 102
9.2 Dynamic friction conditions . . . 103
9.3 Static friction conditions . . . 104
9.4 Impulsive frictional forces . . . 104

10 Dynamic Friction **107**
10.1 Indeterminacy . . . 108
10.2 Inconsistency . . . 114

11 An *NP*-complete Class of Configurations **117**

12 Physical Models **126**
12.1 A model of inconsistency . . . 127
12.2 A model of indeterminacy . . . 129
12.3 Indeterminacy and inconsistency combined . . . 132

12.4 Preferred solutions 133

13 Computing Valid Contact Forces and Impulses **138**
13.1 Defining valid contact impulses 138
13.2 Lemke's algorithm 140
13.3 Continuity of solutions 143

14 Static Friction **144**
14.1 Complexity of static friction 144
14.2 Approximation methods 152
14.2.1 Lötstedt's approximation 152
14.2.2 Dynamic friction approximation 155
14.2.3 Quadratic programming approximation 157

15 Future Directions **160**

A Extremal Point Conditions for Parametric Surfaces **165**

B Termination Conditions of Lemke's Algorithm **168**

C Iterative Solution Methods for Static Friction **173**

D Simulation Examples **178**

Bibliography **185**

List of Figures

2.1 An equality constraint between a particle and a surface. 7
2.2 An inequality constraint between a particle and a surface. 8
2.3 Constraint forces parallel to the surface normal $\nabla C(x(t))$ for an equality constrained problem. 13
2.4 Constraint forces parallel to the surface normal $\nabla C(x(t))$ for an inequality constrained problem. 16

3.1 The computational steps required to compute $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t)$. 23

5.1 The minimum distance points p_a and p_b between two convex surfaces. 46
5.2 Inter-penetration between surfaces. 47
5.3 The definition of the extremal points. 48
5.4 Geometric conditions for the extremal points. 49
5.5 The sweep/sort algorithm. 53
5.6 A coherence-based method of detecting overlaps. 54

6.1 Three configurations in world space and their representation as points in C-space. 59
6.2 The reaction force $F_{reaction}$. 62
6.3 Change of the extremal points as a function of the configuration. . . 64
6.4 C expressed in terms of the extremal points. 65
6.5 Side view of the implicit surfaces F and G expressed explicitly by functions f and g. 69
6.6 Frictionless oscillation of an ellipsoid A on a plane B. 77
6.7 Frictionless oscillation of a more eccentric ellipsoid A on a plane B. 79
6.8 Frictionless oscillation of a superquadric ellipsoid A on a plane B. . 80

10.1 A one contact point configuration between a thin rod A and a fixed base B. 110
10.2 An indeterminate configuration. 113
10.3 A potentially inconsistent configuration. 114

11.1 A possibly inconsistent configuration. 120
11.2 A configuration whose consistency depends on an external force. . . . 121
11.3 A configuration that is consistent if and only if the friction forces on B sum to μS. 122

12.1 A model of indeterminacy. 128
12.2 A configuration with no initial inter-penetration. 130
12.3 A configuration with an initial inter-penetration depth of d_1. 131
12.4 A configuration with a single impulse-free solution. 133
12.5 A configuration with two different non-impulsive solutions at time zero. 136

14.1 Force space. 147
14.2 Line N intersects L and RL. 148
14.3 Line N intersects RL, but not L. 149

15.1 Non-polygonal contact region. 162

D.1 Jack falling down stairs. 179
D.2 Falling superquadric dice. 181
D.3 Collapsing structure. 183

Chapter 1

Introduction

The research problem motivating this thesis has a long and detailed history, having been a driving force behind the development of both continuous mathematics and the high speed digital computer. The problem in question is the problem of *dynamics*: the study of the motions of bodies. Calculus, for instance, was invented by Newton, as he pondered the motion of bodies under the influence of gravity. The first high speed (at the time) digital computer, the Eniac, was built to solve ballistics problems.

The dynamics problems originally posed by Newton were all of an *unconstrained* nature; for example, the motion of two bodies subject only to gravitational forces between them. In contrast, the dynamics problems considered in this thesis are all of a *constrained* nature. In particular, this thesis will consider the problem of simulating the dynamics of rigid bodies subject to *non-penetration constraints*; that is, constraints that require bodies to behave as if they were solid, and avoid inter-penetration.

In order for a simulation to be computable, it is necessary to make simplifying assumptions about the environment being simulated. In order to make simulation

practical, we usually try to find the simplest assumptions that adequately model a particular environment. As the requirements on our simulation are increased (more complex phenomenon, greater accuracy), we must also increase the complexity of our simulation model. It is naturally assumed, therefore, that realistic simulation is hard mainly because it requires a very complex simulation model. However, as this thesis will show, even very simple simulation models yield complex results and questions.

1.1 Thesis concerns and restrictions

In this thesis, we will study the problems involved in simulating the motions of bodies in accordance with relatively simple physical models of bodies' dynamics. Empirical correctness is not a major concern of this thesis. We do not consider the dynamics models used in this thesis to be empirically correct. That is not to say that our dynamics models are *ad hoc*, or deliberately wrong—a more apt description would be to say that our "simple" dynamics models are incomplete descriptions of more "complicated" dynamics models (which are themselves considered simplifications of even more complex models). Given these simple models, our interest lies in the ramifications of using these models as a basis for simulation. The viewpoint taken by this thesis then is not unlike the viewpoint of rational mechanics[25]. Rational mechanics involves logically deducing the dynamics of systems based on a set of axioms. Rational mechanics gives a "correct result" in as much as the axiom set is valid (with respect to the real world) and complete.

Unlike rational mechanics, this thesis will also be concerned with the practical and computational difficulties that arise in trying to perform simulations based on our simple models. If in fact our simple models are "subsets" of more complicated models, then any findings concerning the difficulties of simulations that use our

simple models should serve as a rough lower bound on the simulation difficulties that will be encountered when more complex models are used. Given that the complex models will yield additional difficulties, it is sensible to first discover the problems stemming from the simple models before moving on to work with the more complex models.

A secondary concern of this thesis is offering practical solutions to the problems encountered in actually using our simplified dynamics model for simulation. For some applications (for example, computer animation, virtual reality, or robot motion planning research) a simulator based on the simple dynamics model considered in this thesis is an end-goal in itself. In these cases, it is important to have practical and efficient simulation algorithms at hand. At present, the vast majority of work in constrained dynamics involving non-penetration constraints uses a simple approximation technique that does not completely enforce the non-penetration constraints (even discounting numerical tolerances). In this thesis, techniques that completely enforce non-penetration constraints (within numerical tolerances) are used. These techniques yield significantly better results and have a firmer theoretical basis than the simple approximation methods, but are considerably more complicated.

The bodies considered in this thesis will be restricted to be perfectly rigid bodies. This implies that bodies propagate forces instantly, without delay, and that complex "micromechanical" processes (such as the elastic response during a collision) are replaced with simple "macromechanical" results. A more important restriction concerns the modeling of contact between bodies. Contact between bodies will always be subject to the restriction that it be modeled as occurring at some finite number of points between bodies. Friction, when considered, will be restricted to the Coulomb model of friction. As we shall see, these few simple laws yield simulation problems of considerable depth and complexity.

1.2 Thesis overview

Chapter 2 presents the prototypical problem that characterizes the problem of dynamic simulation with non-penetration constraints.

Chapter 3 gives an overview of the simulation process, in terms of the major computational steps required.

Chapter 4 summarizes previous work, as classified by the major computational steps of simulation defined in Chapter 3.

Chapter 5 presents collision detection algorithms that are well-suited for the type of environment encountered in dynamics simulation.

Chapter 6 develops the local motion constraint that arises between contacting bodies to prevent inter-penetration.

Chapter 7 analyzes the penalty method, which is currently the most popular method for preventing inter-penetration during simulation.

Chapter 8 formulates equations to solve for constraint forces for frictionless systems.

Chapters 9 and 10 introduce a friction model and discuss the complications that arise from its use. Chapter 11 discusses the computational complexity issues that arise from the introduction of friction.

Chapter 12 proposes a physical interpretation of the computational difficulties arising from friction. Chapters 13 and 14 discuss principles and computational methods for simulations with friction.

Chapter 15 concludes this thesis by discussing future directions for simulations with non-penetration constraints.

Chapter 2

A Simple Dynamics Problem with Non-penetration Constraints

In considering the steps needed to solve dynamics problems with non-penetration constraints, it is helpful to have a dynamics problem with non-penetration constraints firmly in mind. The simple dynamics problem presented in this chapter illustrates the important issues of dynamic simulation involving non-penetration constraints. The problem is to simulate the dynamics of a point mass particle in three-space ($\mathbf{R}^3$). The position x of the particle at time t in $\mathbf{R}^3$ is written $x(t)$. The velocity v of the particle at time t is $v(t)$, and the particle is assumed to have unit mass. An arbitrary external force $F(t)$ acts on the particle at time t.

2.1 Unconstrained motion

When there are no constraints on the particle's motion (that is, no obstacles for the particle to encounter), the motion of the particle is simple. Using Newton's

$F = ma$, the differential equation

$$\ddot{x}(t) = F(t) \tag{2–1}$$

describes the particle's motion. In numerical simulation though, this second order differential equation is converted to a coupled first order differential equation by writing

$$\begin{aligned} \dot{x}(t) &= v(t) \\ \dot{v}(t) &= F(t) \end{aligned}$$

or in vector notation,

$$\frac{\mathrm{d}}{\mathrm{dt}} \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ F(t) \end{pmatrix}. \tag{2–2}$$

The variables x and v are called *state variables*, and the collection of x and v is called the *state* of the system. The variable x is a *spatial* variable while v is a *velocity* variable.

2.2 Equality constrained motion

Now suppose that S is a surface in $\mathbf{R}^3$, and supposed the particle is constrained to always lie on S. Let S be modeled implicitly by a scalar function C; that is, a point p lies on S if and only if $C(p) = 0$. Using C, the constraint on the particle can be written

$$C(x(t)) = 0. \tag{2–3}$$

Since this constraint can be written as an equality, this is an example of an *equality constrained* dynamics problem (Figure 2.1).

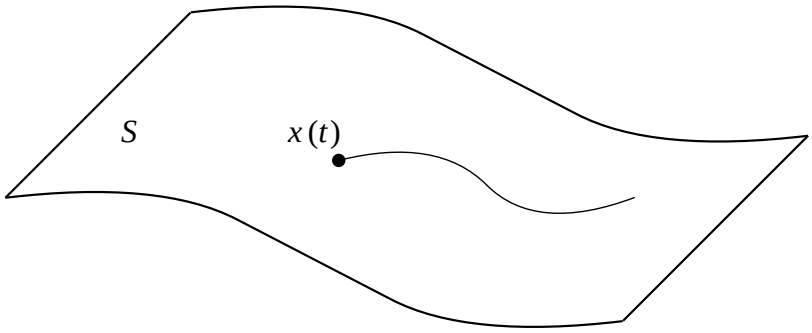

Figure 2.1: An equality constraint between a particle and a surface. The particle, constrained by $C(x(t)) = 0$, must always lie on the surface S.

2.3 Inequality constrained motion

Now suppose that the constraint of the previous problem is relaxed, so that the particle is constrained to lie either on or "above" S (Figure 2.2). If points p lying above S satisfy $C(p) > 0$, then the constraint that the particle lie on or above S can be written

$$C(x(t)) \geq 0. \tag{2–4}$$

Since the constraint requires an inequality, this is an example of an *inequality constrained* dynamics problem.

The inequality constrained dynamics problem is a superset of the unconstrained and equality constrained dynamics problems, and is thus the most difficult problem of the three. Clearly, the inequality constrained dynamics problem subsumes the unconstrained dynamics problem; if at time t the particle satisfies $C(x(t)) > 0$, then for some period of time following t, the motion of the particle can be regarded as unconstrained. Similarly, the equality constrained problem is a special case of

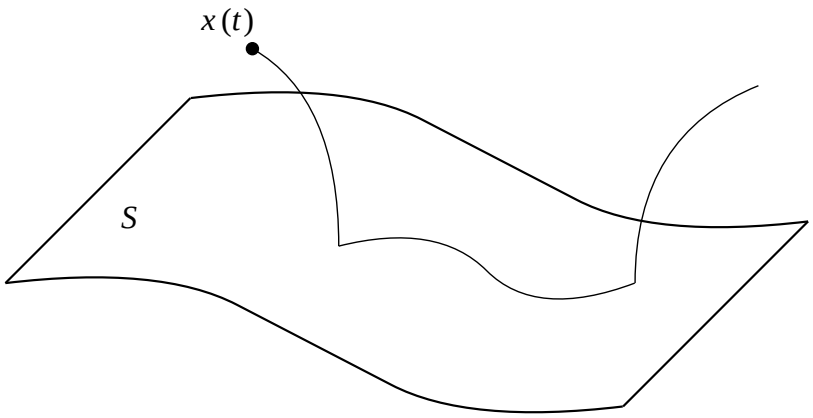

Figure 2.2: An inequality constraint between a particle and a surface. The particle, constrained by $C(x(t)) \geq 0$, can move either on or "above" the surface S.

the inequality constrained problem—the equality constraint $C(x(t)) = 0$ can be replaced by the constraints $C(x(t)) \geq 0$ and $-C(x(t)) \geq 0$.

2.4 The penalty method

Constrained dynamics problems can be very difficult to solve. A very common method for solving a problem with constraints is the *penalty method*. The penalty method for constrained dynamics problems is based on a numerical solution method for *constrained optimization* problems. In both fields, the penalty method converts a constrained problem to an unconstrained problem where deviation from the constraint is penalized; that is, in the new problem, satisfaction of the constraint is encouraged, but not strictly enforced.

In optimization, a typical equality constrained problem such as

$$\text{minimize } f(z) \quad \text{such that} \quad g(z) = 0 \tag{2–5}$$

can be rewritten as an unconstrained problem

$$\text{minimize } f(z) + kg(z)^2 \quad \text{as } k \to \infty. \tag{2–6}$$

The term $kg(z)^2$ is called the *penalty function*. The idea is that as k grows larger, potential solutions for z must make $g(z)^2$ smaller, to minimize Equation (2–6). In the limit, as k goes to infinity, the solution of Equation (2–6) must satisfy $g(z) = 0$ while minimizing $f(z)$. In practice, z is obtained by solving Equation (2–6) for a series of increasing values of k until the series of solutions converges (within numerical tolerance) to a limit. Although the method has a theoretically firm basis, in practice, it is not a very robust numerical method. The main problem is that as k grows, Equation (2–6) can become very poorly conditioned and difficult to solve[14]. The main attraction of the method is that it a very simple way of turning a constrained problem into an unconstrained problem.

The equality constrained particle/surface problem can be converted to an unconstrained dynamics problem in a similar fashion. The idea is that the particle is not explicitly constrained to lie on the surface. However, if the particle drifts off the surface, a penalty force acts on the particle, in a direction normal to the surface, so as to pull the particle back to the surface. Typically, the penalty force is modeled as a linear spring force; that is, the penalty force pulls the particle back towards the surface with a a strength equal to some constant k times the distance of the particle from the surface. If we let $p(x(t))$ denote the closest point on the surface to $x(t)$, then the penalty force is[1]

$$-k\left(x(t) - p(x(t))\right).$$

[1] If the surface is a plane, the closest point on the plane to the particle is unique. Otherwise, the closest point on the surface to the particle is unique as long as the particle remains in some neighborhood containing the surface; the extent of this neighborhood depends on the curvature of the surface. Since the penalty method is intended to keep the particle close to the surface, the closest point on the surface to the particle is always assumed to be well-defined.

Note that as $x(t)$ drifts farther from the surface, the force $-k\left(x(t) - p(x(t))\right)$ grows larger in magnitude. The introduction of the penalty force reduces the dynamics problem to simply an evaluation of the ordinary differential equation

$$\frac{\mathrm{d}}{\mathrm{dt}} \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ -k\left(x(t) - p(x(t))\right) + F(t) \end{pmatrix}. \tag{2–7}$$

If the penalty method for dynamics were to completely emulate the penalty method for constrained optimization, the simulation would be repeated with increasing values of k until the behavior of the particle approached a limit. However, the penalty method, as used by dynamics, chooses a single value for k. If the value chosen for k is too small, it may do an inadequate job of enforcing the constraint; in this case, the simulation would be rerun with a higher value of k. Unfortunately, as in the optimization problem, ill-conditioning occurs as k grows larger, in the guise of "stiffness" for the differential equation of motion. Stiff differential equations are expensive and difficult to solve.

The penalty method can also be used for the inequality constrained particle/surface problem. The modification is straightforward; whenever the particle lies below the surface, a penalty force acts upwards on the particle, as in the equality constrained case. However, when the particle lies above the surface, no penalty force acts on the particle.

For neither the equality constrained problem nor the inequality constrained problem is it obvious that the penalty method must give a correct solution, given a sufficiently large value of k. It is clear however that except for a limited class of simulations, the penalty method is an unsuitable method, because of the ill-conditioning of the differential equations of motion. Further analysis of the penalty method will be deferred to Chapter 7.

2.5 Exact solution methods

The penalty method of the previous section involves a formulation of the problem in which constraints are only approximately satisfied. A completely different viewpoint is taken in this section, in that constraints are required to be satisfied "exactly" (within numerical tolerances).

2.5.1 The equality constrained problem

The equality constrained problem of the particle that must remain in contact with the surface S can be solved in two ways. The first solution method involves converting the constrained problem to an unconstrained problem—essentially the method of Lagrangian dynamics. This can be done if a parametric representation for S can be found.

Otherwise, the problem can be solved by introducing a *constraint force* $F_c(t)$ into the problem. The role of the constraint force is to act on the particle so that it remains in contact with the surface. Additionally, the constraint force must be *workless*; that is, the work done on the particle by the constraint force must always be zero. For the particle/surface example, this requirement forces $F_c(t)$ to act on the particle in a direction normal to the surface S at $x(t)$. Thus, the constraint force's responsibility is to adjust the acceleration of the particle normal to the surface, so that the particle will remain on the surface. The normal to S at the point $x(t)$ is

the vector $\nabla C(x(t))$ where[2]

$$\nabla C(x(t)) = \begin{pmatrix} \frac{\partial C}{\partial x}\big(x(t)\big) \\ \frac{\partial C}{\partial y}\big(x(t)\big) \\ \frac{\partial C}{\partial z}\big(x(t)\big) \end{pmatrix}.$$

Then $F_c(t)$ can be written $F_c(t) = \lambda(t)\nabla C(x(t))$ where $\lambda(t)$ is an unknown scalar. The scalar $\lambda(t)$ can be either positive or negative. In Figure 2.3a, the constraint force $F_c(t)$ acts to oppose a large external force $F(t)$. The constraint force acts in the same direction as $\nabla C(x(t))$; in this case, $\lambda(t) > 0$. In Figure 2.3b, the external force $F(t)$ now pushes the particle away from the surface in the $\nabla C(x(t))$ direction. In this case, the constraint force must act opposite $\nabla C(x(t))$; hence $\lambda(t) < 0$.

The equation of motion for the particle then becomes

$$\ddot{x}(t) = F(t) + F_c(t) = F(t) + \lambda(t)\nabla C(x(t)). \tag{2–8}$$

How is $\lambda(t)$ computed?

In equality constrained dynamics problems, it is assumed that the constraint is initially met; that is, at time zero,

$$C(x(0)) = 0. \tag{2–9}$$

Additionally,

$$\dot{C}(x(0)) = \frac{\mathrm{d}}{\mathrm{dt}} C(x(0)) = \nabla C(x(0)) \cdot \dot{x}(0) \tag{2–10}$$

must also be zero as well. If it is not, then the particle has a velocity that is not completely tangential to S at the point $x(0)$; this situation will require an *impulsive* constraint force, and will be dealt with shortly. If however both $C(x(t))$

[2] It is assumed that the function C modeling S is chosen such that $\nabla C(p)$ does not vanish on S.

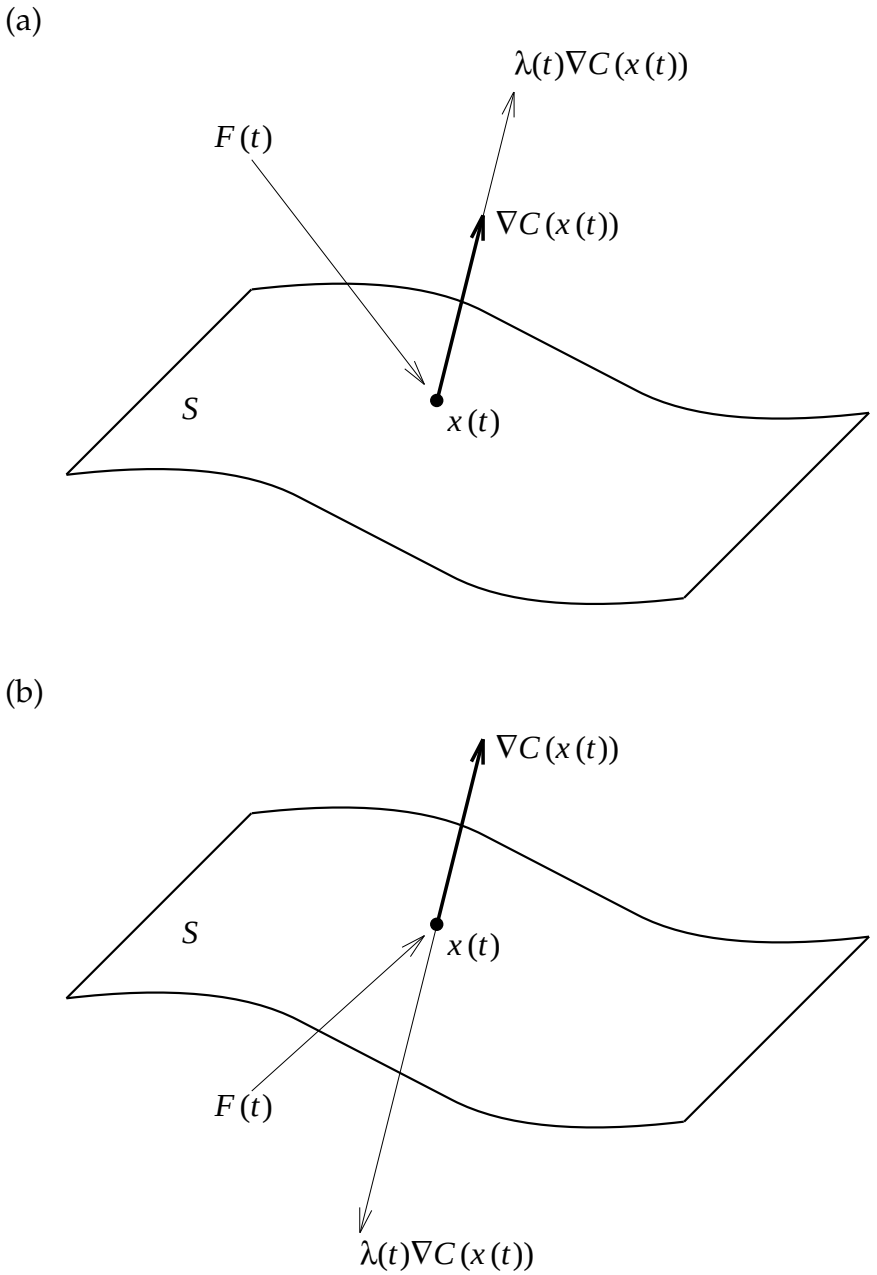

Figure 2.3: Constraint forces parallel to the surface normal $\nabla C(x(t))$ for an equality constrained problem. (a) To maintain the particle/surface constraint, $\lambda(t) > 0$ so that the constraint force $F_c(t) = \lambda(t)\nabla C(x(t))$ pushes the particle in the positive $\nabla C(x(t))$ direction. (b) The constraint force $F_c(t)$ must be applied opposite to $\nabla C(x(t))$; thus $\lambda(t) < 0$.

and $\dot{C}(x(t))$ are zero for $t = 0$, then both quantities will remain zero as long as $\ddot{C}(x(t)) = \frac{\mathrm{d}^2}{\mathrm{dt}^2} C(x(t))$ is zero. Intuitively, $C(x(t))$ is a measure of the particle's displacement normal to S, and $\dot{C}(x(t))$ is a measure of the particle's velocity normal to S. Thus, $\ddot{C}(x(t))$ is a measure of the particle's acceleration normal to S. The constraint force $\lambda(t)\nabla C(x(t))$ is chosen so that $\ddot{C}(x(t))$ is always zero.

Starting from $\dot{C}(x(t)) = \nabla C(x(t)) \cdot \dot{x}(t)$ and taking its derivative,

$$\begin{aligned} \ddot{C}(x(t)) &= \frac{\mathrm{d}}{\mathrm{dt}} \left(\nabla C(x(t)) \cdot \dot{x}(t)\right) \\ &= \left(\frac{\mathrm{d}}{\mathrm{dt}} \nabla C(x(t))\right) \cdot \dot{x}(t) + \nabla C(x(t)) \cdot \left(\frac{\mathrm{d}}{\mathrm{dt}} \dot{x}(t)\right) \qquad (2\text{–}11) \\ &= \dot{x}(t) \cdot \left(\nabla^2 C(x(t)) \dot{x}(t)\right) + \nabla C(x(t)) \cdot \ddot{x}(t) \end{aligned}$$

where the term $\nabla^2 C(x(t))$ is the matrix of second partial derivatives of C (that is, the curvature of the surface), evaluated at the point $x(t)$. Using Equation (2–8),

$$\begin{aligned} \ddot{C}(x(t)) &= \left(\dot{x}(t) \cdot \left(\nabla^2 C(x(t)) \dot{x}(t)\right)\right) + \left(\nabla C(x(t)) \cdot \ddot{x}(t)\right) \\ &= \left(\dot{x}(t) \cdot \left(\nabla^2 C(x(t)) \dot{x}(t)\right)\right) + \left(\nabla C(x(t)) \cdot \left(F(t) + \lambda(t) \nabla C(x(t))\right)\right) \\ &= \dot{x}(t) \cdot \left(\nabla^2 C(x(t)) \dot{x}(t)\right) + \nabla C(x(t)) \cdot F(t) \qquad (2\text{–}12) \\ &\quad + \lambda(t) \nabla C(x(t)) \cdot \nabla C(x(t)). \end{aligned}$$

Setting $\ddot{C}(x(t))$ to zero and solving for $\lambda(t)$,

$$\lambda(t) = -\frac{\dot{x}(t) \cdot (\nabla^2 C(x(t)) \dot{x}(t)) + \nabla C(x(t)) \cdot F(t)}{\nabla C(x(t)) \cdot \nabla C(x(t))}. \qquad (2\text{–}13)$$

Having found $\lambda(t)$, Equation (2–8) can be solved numerically, and the motion of the particle will satisfy the equality constraint.

2.5.2 The inequality constrained problem

The solution to the inequality case is only slightly more complicated for this simple example. Suppose that initially the particle lies above the surface S, and does not come into contact with the surface until time t_1. During the time interval zero to t_1 the constraint is said to be *inactive* and the motion of the particle satisfies

$$\ddot{x}(t) = F(t).$$

The first problem then is to determine t_1, that is, to determine when the particle contacts the surface. This is the problem of *collision detection.*

After the particle contacts the surface at time t_1, the constraint $C(x(t)) \geq 0$ becomes active. To prevent the particle from moving below the surface, a constraint force must act. In what follows, it is assumed that $\nabla C(x(t))$ points "upwards" (towards the region of space that the particle is permitted to move in) as in Figure 2.4. As in the equality constrained problem, the constraint force is assumed to do no work on the particle; so again, the constraint force can be written as $F_c(t) = \lambda(t)\nabla C(x(t))$. Unlike the equality constrained problem, the constraint force must be directed away from the surface, in the $\nabla C(x(t))$ direction. This means that $\lambda(t)$ must satisfy $\lambda(t) \geq 0$. In Figure 2.4a, where the external force $F(t)$ pushes the particle downwards, the constraint force $\lambda\nabla C(x(t))$ is exactly the same as in Figure 2.3a. However, when the external force $F(t)$ of Figure 2.4b acts on the particle, *no* constraint force acts (that is, $\lambda(t)$ is zero), and the particle breaks contact with S, accelerating upwards. If $\lambda(t)$ was allowed to be negative, as in Figure 2.3b, the particle, once in contact with the surface, could be held to the surface for all time by computing a value for $\lambda(t)$ as in the equality constrained problem.

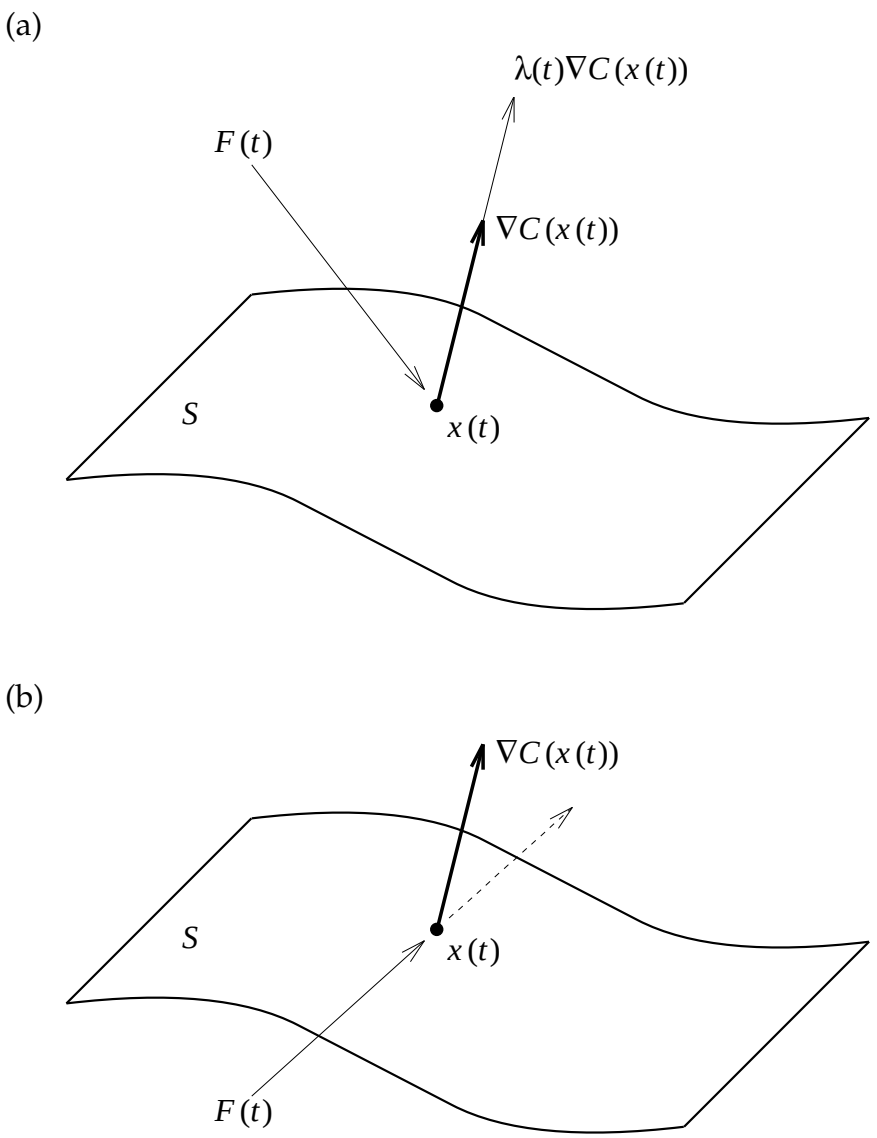

Figure 2.4: Constraint forces parallel to the surface normal $\nabla C(x(t))$ for an inequality constrained problem. (a) A constraint force $F_c(t) = \lambda(t)\nabla C(x(t))$ with $\lambda(t) > 0$ acts to prevent the particle from accelerating downwards. (b) The external force $F(t)$ accelerates the particle upwards, and contact is broken. No constraint force acts on the particle.

First derivative considerations

If at time t_1 (when the particle first contacts the surface)

$$\dot{C}(x(t_1)) = \nabla C(x(t_1)) \cdot \dot{x}(t_1) = \nabla C(x(t_1)) \cdot v(t_1) < 0, \tag{2–14}$$

then the particle is said to have *collided* with the surface. To prevent the particle from moving below the surface, the velocity $v(t)$ must undergo a discontinuity at time t_1; that is the only means of completely enforcing the constraint. The discontinuity is said to be the result of an *impulsive* constraint force that acts normal to the surface at the point $x(t_1)$. An impulsive force has the dimensions of force times time, or equivalently mass times velocity, and produces a discontinuity in the velocity. Impulsive forces are the limiting effect of large forces acting over small time intervals, as the force strength goes to infinity and the time interval goes to zero. Thus, the impulsive constraint force will also be normal to the surface, and can be expressed as $j\nabla C(x(t_1))$ where j is a positive scalar.

Because the particle and the surface are regarded as rigid bodies, the strength of the impulsive force is found by using the empirical relation

$$v_n^+ = -\epsilon v_n^- . \tag{2–15}$$

In this relation, v_n^+ is the velocity of the particle normal to the surface after the impulsive constraint force is applied, and v_n^- is the normal velocity prior to the collision. The scalar $0 \leq \epsilon \leq 1$ is called the *coefficient of restitution* and depends on the material properties of the particle and the surface.

Since the particle has unit mass, its velocity $v^+(t_1)$ after application of the impulse $j\nabla C(x(t_1))$ is

$$v^+(t_1) = v(t_1) + j\nabla C(x(t_1)). \tag{2–16}$$

(This equation can be formally derived by defining $v^+(t_1) = \lim_{\Delta t \to 0} v(t_1 + \Delta t)$ when a force $\lambda \nabla C(x(t))$ acts on the particle from time t_1 to time $t_1 + \Delta t$, with λ

a scalar, and $\lim_{\Delta t \to 0} \lambda \Delta t = j$.) The normal velocities v_n^- and v_n^+ are

$$v_n^- = \nabla C(x(t_1)) \cdot v(t_1) \qquad \text{and} \qquad v_n^+ = \nabla C(x(t_1)) \cdot v^+(t_1). \tag{2–17}$$

Substituting into Equation (2–15),

$$\nabla C(x(t_1)) \cdot v^+(t_1) = -\epsilon \nabla C(x(t_1)) \cdot v(t_1) \tag{2–18}$$

or

$$\nabla C(x(t_1)) \cdot (v(t_1) + j \nabla C(x(t_1))) = -\epsilon \nabla C(x(t_1)) \cdot v(t_1). \tag{2–19}$$

Solving for j,

$$\begin{aligned} j &= \frac{-\epsilon \nabla C(x(t_1)) \cdot v(t_1) - \nabla C(x(t_1)) \cdot v(t_1)}{\nabla C(x(t_1)) \cdot \nabla C(x(t_1))} \\ &= -\frac{(\epsilon + 1) \nabla C(x(t_1)) \cdot v(t_1)}{\nabla C(x(t_1)) \cdot \nabla C(x(t_1))} \\ &= -\frac{(\epsilon + 1) \dot{C}(x(t_1))}{\nabla C(x(t_1)) \cdot \nabla C(x(t_1))}. \end{aligned} \tag{2–20}$$

Note that since $\dot{C}(x(t_1)) < 0$ and $\nabla C(x(t_1)) \cdot \nabla C(x(t_1)) > 0$, then $j > 0$. This indicates that the constraint impulse acts upwards on the particle, as expected.

If $\epsilon > 0$, then the particle will have a velocity along $\nabla C(x(t_1))$ after the collision; the particle has bounced off the surface, and is now moving away from the surface. The constraint changes back to being inactive, and the motion of the particle is considered unconstrained until such time as it contacts the surface again. Note that this entire process happens over a zero length time interval; thus, the motion of the particle is piecewise continuous.

Otherwise, if $\epsilon = 0$, then the particle has no velocity normal to the surface after the collision. In this case, the constraint stays active, and a regular, non-impulsive constraint force can be considered.

Second derivative considerations

The case when $\dot{C}(x(t)) < 0$ has just been discussed. If $\dot{C}(x(t)) > 0$, the particle has a velocity directed away from the surface, and the constraint immediately becomes inactive after time t. Thus, in considering the constraint forces that prevent the particle from moving below the surface at time t, we may assume

$$C(x(t)) = \dot{C}(x(t)) = 0. \tag{2–21}$$

In the equality constrained problem, $C(x(t)) = 0$ was maintained by choosing $\lambda(t)$ such that $\ddot{C}(x(t)) = 0$. For the inequality constrained problem, $C(x(t)) \geq 0$ is maintained by choosing $\lambda(t)$ so that $\ddot{C}(x(t)) \geq 0$, subject to the condition $\lambda(t) \geq 0$. Substituting from Equation (2–12) into $\ddot{C}(x(t)) \geq 0$, we obtain

$$\begin{aligned} \ddot{C}(x(t)) &= \dot{x}(t) \cdot \left(\nabla^2 C(x(t))\dot{x}(t)\right) + \nabla C(x(t)) \cdot F(t) \\ &\quad + \lambda(t)\nabla C(x(t)) \cdot \nabla C(x(t)) \geq 0. \end{aligned} \tag{2–22}$$

Since $\nabla C(x(t)) \cdot \nabla C(x(t))$ is always positive, $\ddot{C}(x(t)) \geq 0$ is equivalent to

$$\lambda(t) \geq -\frac{\dot{x}(t) \cdot (\nabla^2 C(x(t))\dot{x}(t)) + \nabla C(x(t)) \cdot F(t)}{\nabla C(x(t)) \cdot \nabla C(x(t))}. \tag{2–23}$$

Equation (2–23) places only a lower bound on $\lambda(t)$, and does not specify $\lambda(t)$ uniquely. However, a reasonable restriction to impose on $\lambda(t)$ is that it be zero whenever $\ddot{C}(x(t)) > 0$. This is sensible because if $\ddot{C}(x(t)) > 0$ then $C(x(t))$ is an increasing function and the particle will move away from the surface immediately after time zero. When this happens, the constraint becomes inactive and the constraint force is non-existent; or equivalently, $\lambda(t)$ is zero. This restriction can be written very simply as

$$\lambda(t)\ddot{C}(x(t)) = 0 \tag{2–24}$$

which, along with $\ddot{C}(x(t)) \geq 0$ and $\lambda(t) \geq 0$, asserts that either $\ddot{C}(x(t)) > 0$ and $\lambda(t) = 0$, or $\ddot{C}(x(t)) = 0$.

With this added restriction, $\lambda(t)$ is computed very easily. If $\lambda(t)$'s lower bound (Equation (2–23)) is positive, then setting $\lambda(t)$ equal to its lower bound results in $\lambda(t) > 0$ and $\ddot{C}(x(t)) = 0$; otherwise, $\lambda(t)$'s lower bound is negative. Since the denominator of Equation (2–23) is always positive, it must be that

$$\dot{x}(t) \cdot (\nabla^2 C(x(t))\dot{x}(t)) + \nabla C(x(t)) \cdot F(t) > 0. \tag{2–25}$$

Setting $\lambda(t) = 0$ satisfies $\lambda(t) \geq 0$ and $\lambda(t)\ddot{C}(x(t)) = 0$. Using Equation (2–22), $\ddot{C}(x(t))$ satisfies

$$\begin{aligned}
\ddot{C}(x(t)) &= \dot{x}(t) \cdot \Big(\nabla^2 C(x(t))\dot{x}(t)\Big) + \nabla C(x(t)) \cdot \ddot{x}(t) \\
&= \dot{x}(t) \cdot \Big(\nabla^2 C(x(t))\dot{x}(t)\Big) + \nabla C(x(t)) \cdot \Big(F(t) + \lambda(t)\nabla C(x(t))\Big) \\
&= \dot{x}(t) \cdot \Big(\nabla^2 C(x(t))\dot{x}(t)\Big) + \nabla C(x(t)) \cdot F(t) > 0.
\end{aligned}$$

Note that during the period of time that $\ddot{C}(x(t)) = 0$, the value of $\lambda(t)$ is the same as if the inequality constraint was in fact an equality constraint; during this time, the particle's behavior is the same as if the particle had been constrained to remain on the surface. However, when in the equality constrained problem $\lambda(t)$ would have decreased below zero, in the inequality constrained problem $\lambda(t)$ stops decreasing at zero and the particle breaks contact with the surface.

Chapter 3

Simulation Overview

Having described a simple, yet complete, problem with non-penetration constraints, the overall structure of a simulation algorithm for the problem is presented, along with the extensions needed to simulate an arbitrary system of rigid bodies.

Overall, the simulation process can be described as nothing more than the numerical solution to a first order ordinary differential equation (ODE). A first order ODE has the form $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t) = f(t, \mathbf{Y}(t))$. The ODE for a system of arbitrarily many rigid bodies is given in Section 3.1. The vector $\mathbf{Y}(t)$ is called the *state* of the system, and $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t) = f(t, \mathbf{Y}(t))$ is called the *state derivative*. Numerical methods for solving such equations are well known. The simulator discussed in this thesis uses an *explicit* solution method (as opposed to an *implicit* solution method) for solving the ODE. (Numerical solution techniques for ordinary differential equations are discussed by Shampine and Gordon[47].) An explicit solution method requires a value for $\mathbf{Y}(0)$, which is called the *initial state* of the system, and a time t_{end} for which the solution method is to compute $\mathbf{Y}(t_{end})$. In computing $\mathbf{Y}(t_{end})$, the solution method will need to be given values for the state derivative at various times from zero to t_{end}. The solution method is completely responsible for choosing values

of t at which to compute the state derivative, and for computing $\mathbf{Y}(t_{end})$ based on the computed state derivatives.

Thus, from a mathematical perspective, the work of a simulation lies "merely" in the task of periodically computing the state derivative. For each value of t for which the state derivative $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t)$ is to be computed, the sequence of steps in Figure 3.1 is performed. The meanings of the steps in Figure 3.1 are explained below. The computation of the state derivative will sometimes be referred to as a *time step*; successive time steps refers to successive computations of the state derivative for different values of t. The *interval* between two time steps is the difference in the value of t between the time steps.

3.1 Updating the current state

When the numerical ODE solver requests a value for $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t)$ at some time t_0, it begins by divulging the *current state* of the system; that is, its estimation of the value of $\mathbf{Y}(t_0)$.

For the particle/surface problem, the current state is simply the vector containing $x(t_0)$ and $v(t_0)$. For a collection of n rigid bodies, the current state is somewhat more complicated, but conceptually the same. The ODE for a collection of n rigid

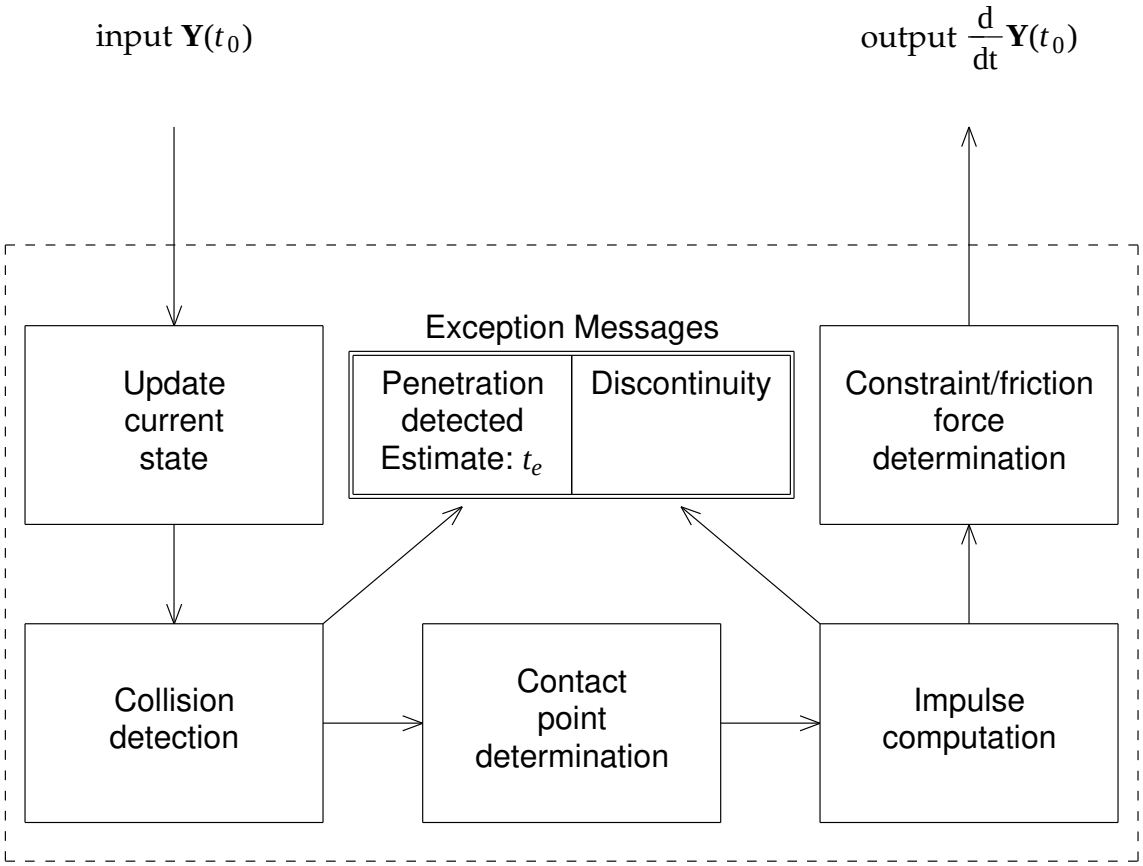

Figure 3.1: The computational steps required to compute $\frac{d}{dt}\mathbf{Y}(t)$. During the computation, various "exceptions" can be communicated to the ODE solver.

bodies has the form

$$\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t) = \frac{\mathrm{d}}{\mathrm{dt}}\begin{pmatrix} c_1(t) \\ q_1(t) \\ v_1(t) \\ \omega_1(t) \\ \vdots \\ c_n(t) \\ q_n(t) \\ v_n(t) \\ \omega_n(t) \end{pmatrix} = \begin{pmatrix} v_1(t) \\ \frac{1}{2}\omega_1(t)q_1(t) \\ F_1(t)/m_1 \\ I_1^{-1}\tau_1(t) \\ \vdots \\ v_n(t) \\ \frac{1}{2}\omega_n(t)q_n(t) \\ F_n(t)/m_n \\ I_n^{-1}\tau_n(t) \end{pmatrix} \tag{3–1}$$

where $c_i(t)$ is the position of the ith body's center of mass, $q_i(t)$ is the unit quaternion representation of the ith body's orientation[21,48], $v_i(t)$ and $\omega_i(t)$ are the linear and angular velocities of the ith body, $F_i(t)$ and $\tau_i(t)$ are the total force and torque acting on the ith body, and m_i and $I_i(t)$ are the ith body's mass and inertia tensor. The term $\omega_i(t)q_i(t)$ denotes quaternion multiplication between $0 + \omega_{xi}(t)\mathbf{i} + \omega_{y_i}(t)\mathbf{j} + \omega_{zi}(t)\mathbf{k}$ and $q_i(t)$.

The values of all the position and velocity variables of the bodies are taken from $\mathbf{Y}(t_0)$ during the computation the derivative state at time t_0.

3.2 Collision detection

In the particle/surface problem, the motion of the particle is treated as unconstrained whenever the particle lies above the surface. Suppose that at some time t_0 the particle is above, but near the surface, and has a velocity downwards (towards the surface). Based on the history of the particle's velocity and acceleration up to time t_0, the ODE solver computes a position for the particle at some time $t_0 + \Delta t$. However, because the ODE solver has no knowledge of the non-penetration

constraint between the particle and the surface, it does not know that the particle cannot continue to move downwards indefinitely. The ODE computes a state $\mathbf{Y}(t_0 + \Delta t)$ for the particle such that the particle lies below the surface; clearly, this is incorrect. The ODE solver then requests the value of $f(t_0 + \Delta t, \mathbf{Y}(t_0 + \Delta t))$, unaware of the fact that $\mathbf{Y}(t_0 + \Delta t)$ is an *illegal* (and hence incorrect) value for the state at time $t_0 + \Delta t$.

It is the collision detection step's responsibility to examine the current state (or rather the *proposed* current state). If the state is illegal, that is, if the non-penetration constraint has been violated, then at some time t_c lying between t_0 and $t_0 + \Delta t$, a collision must have occurred. The collision detector makes some estimate t_e of t_c, informs the ODE solver that the proposed state $\mathbf{Y}(t_0 + \Delta t)$ is illegal, and gives the estimated value t_e. The value of $f(t_0 + \Delta t, \mathbf{Y}(t_0 + \Delta t))$ is undefined since the proposed state $\mathbf{Y}(t_0 + \Delta t)$ is illegal, and computation of $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t_0 + \Delta t)$ is aborted. The ODE solver proceeds forward from time t_0 and computes a proposed state for $\mathbf{Y}(t_0 + t_e)$.

If $t_e < t_c$, the proposed state $\mathbf{Y}(t_0 + t_e)$ is legal and it is now known that $t_e < t_c < t_0 + \Delta t$. Based on information from the state $\mathbf{Y}(t_e)$, it may be possible to produce a better estimate for t_c, which the solver can aim for. Failing that, the solver can at least bisect the time interval from t_e to $t_0 + \Delta t$ by estimating t_c to be $\frac{1}{2}(t_e + t_0 + \Delta t)$.

Conversely, if $t_c < t_e$, then the proposed state $\mathbf{Y}(t_e)$ is illegal, and the solver knows that $t_0 < t_c < t_e$. Again, either a better prediction for t_c can be made based on $\mathbf{Y}(t_e)$, or the time interval can be bisected. Eventually, the solver finds t_c to within a suitable numerical precision; that is, a state is reached for which the non-penetration constraint is satisfied within numerical tolerances and the velocities of bodies are such that a collision is occurring.

One potential difficulty lies in the fact that if two successive states $\mathbf{Y}(t_0)$ and

$\mathbf{Y}(t_0 + \Delta t)$ are determined to be legal, it is still possible that a collision might have occurred somewhere between these two states. This could happen if one body passed completely through another between time t_0 and time $t_0 + \Delta t$. Consider for example simulating the motion of a bullet shot through a thin pane of glass. In this case, the odds are against the solver requesting $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t)$ at a value of t for which the bullet inter-penetrates the glass. However, for reasonably complicated simulations, the numerical considerations in solving the differential equations of motion limit the interval between time steps and the problem arises infrequently. For this reason, we choose not to treat the case where a body passes completely through another body in one time step. Possible solutions to this problem are using some type of 4D space-time volume algorithms to detect collisions, or simply limiting the maximum interval between time steps.

3.3 Contact point determination

Once a legal state $\mathbf{Y}(t_0)$ has been achieved, the constraint forces at time t_0 must be computed. The first step in that computation is determining *where* the constraint forces can arise. For the particle/surface problem, this determination is simple; if the particle lies on the surface (within some numerical tolerance), a *contact point* is said to exist between the particle and the surface. The location of this contact point is simply the position of the particle. If a constraint impulse or a constraint force is needed, it will act between the particle and the surface at the contact point. For systems of rigid bodies, there may be arbitrarily many points of contact between the bodies, and a constraint force may act at each point. Thus, it is necessary to first determine all the contact points between the bodies before giving further consideration to the constraint forces.

In this thesis, contact between bodies has been restricted to the case when it

can be modeled as occurring at some finite number of points between bodies. This restriction does not apply to contact between polyhedral bodies. For polyhedral bodies, contact areas are always polygonal (considering single points and line segments as degenerate polygons). By imposing a non-penetration constraint at each vertex of the polygonal contact area, inter-penetration is prevented over the entire polygonal contact area. Additionally, any distribution of normal forces over the contact area is equivalent to some assignment of normal forces acting only at the vertices of the contact region. Thus, the definition of a contact point for the case of polyhedral contact is limited to the vertices of the contact region.

Although the collision detection and contact point determination steps are listed separately in Figure 3.1, the computation of each overlaps. As bodies are checked to see if non-penetration constraints have been violated, contact points are determined at the same time. Both steps can share a significant amount of computation, and performing them separately would be inefficient. When a proposed state is found to be illegal, the work done so far in determining contact points is wasted; but it would be much more wasteful to implement the two steps completely separately.

3.4 Impulse computation

Once the contact points have been determined, the relative velocity at each contact point can be computed. If the velocity at a contact point is such that the contact is breaking, no non-penetration constraint need be formulated at the contact point; equivalently, the non-penetration constraint at the contact point is inactive.

If the velocity at a contact point is such that a collision is taking place, an impulsive force will arise. If it is assumed that no two collisions occur at exactly the same time, the calculation of the impulse, for frictionless surfaces, is exactly as described in Section 2.5. Methods for dealing with simultaneous impulses, and

impulses involving friction are summarized in Chapter 4.

Since impulsive forces cause discontinuities in the velocity variables $v_i(t)$ and $\omega_i(t)$ of the rigid bodies, special care must be taken when impulses are applied. Numerical ODE solvers assume that the function $\mathbf{Y}(t)$ is smooth. Since $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t) = f(t, \mathbf{Y}(t))$ is only piecewise continuous, it must be treated as such. Accordingly, when an impulse is applied at time t_0, new values for the affected $v_i(t)$ and $\omega_i(t)$ must be computed, and the ODE solver must be informed that a discontinuity has been encountered. The ODE solver is then restarted with a new state $Y^+(t_0)$, where $Y^+(t_0)$ contains the new values of the affected $v_i(t)$ and $\omega_i(t)$. Note that if the ODE solver begins by requesting a value for $\frac{\mathrm{d}}{\mathrm{dt}}Y^+(t_0)$, the collision detection and contact point determination information computed for $\mathbf{Y}(t_0)$ can be used in computing $\frac{\mathrm{d}}{\mathrm{dt}}Y^+(t_0)$, since $\mathbf{Y}(t_0)$ and $Y^+(t_0)$ represent the same geometric positionings of bodies.

3.5 Constraint/friction force determination

This step is the easiest to describe, but the most difficult to perform. At this point in the computation of $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t_0)$, all contact points with active non-penetration constraints have been identified. Since any constraints requiring impulsive constraint forces have already been taken care of, all that remains is the formulation and solution of a system of equations for the unknown constraint forces. While constraint forces always act normally between contacting bodies, contacts involving friction give rise to forces that act tangentially between bodies, to oppose slipping. For contacts with friction, a system of equations for both the unknown constraint forces and tangential friction forces needs to be formulated and solved.

Once the constraint forces and the friction forces have been computed, the right-hand-side of Equation (3–1) is trivially computed, and the computation of $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t_0)$ is complete. The ODE solver computes a new (proposed) state $\mathbf{Y}(t_0 + \Delta t)$, and computation of $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t_0 + \Delta t)$ begins.

Chapter 4

Summary of Previous Work

In this chapter, previous work relating to the various simulation steps outlined in Chapter 3 is summarized.

4.1 Collision/contact determination

The two problems of collision detection and contact point determination are very similar, and can really be considered a single problem, which we shall call the *collision/contact determination* problem. There is an immense amount of literature concerning this problem, but the majority of collision/contact determination algorithms fall into one of two groups. First, the collision/contact determination problem from time t_0 to t_1 can be viewed as a single, continuous function of time. Given this viewpoint, the basic problem to be solved is "at what time, and where, do bodies first come into contact?" Second, the problem can be considered discretely, at a sequence of time values $t_0 < t_0 + \Delta t_1 < t_0 + \Delta t_2 < \cdots < t_0 + \Delta t_n < t_1$. In this viewpoint, the basic problem is "given the positions of bodies at time $t_0 + \Delta t_i$, where do bodies inter-penetrate and contact each other?"

The first approach, which could be called the "continuum view", presupposes

a specified motion of bodies over some time interval. Examples include Canny[6], who describes an algorithm for determining the first collision between rigid polyhedral objects with constant angular velocity. Using the constant angular velocity assumption, Canny reduces the problem of determining the first instant of collision to the problem of finding roots of (relatively low order) polynomials. Gilbert and Hong[18] relax the problem of collision detection to include arbitrary trajectories of rigid convex polyhedra. Since no closed-form solution exists for time at which the the first intersection occurs, an iterative numerical method is used to determine this time. Their algorithm is based in part on previous work by Gilbert, Johnson, and Keerthi[19]. Von Herzen, Barr, and Zatz[53] describe an algorithm that determines the first collision between parametrically defined time-dependent curved surfaces. In this work, the motion of the bodies is not necessarily rigid (that is, the actual shape of the surfaces may vary over time). Unfortunately, algorithms of the above nature cannot be used in the simulation problem as described in Chapter 3. All of the above algorithms presuppose a specified motion for the bodies over time; however, such a path is exactly what the simulator is trying to compute, which means we cannot make use of these sorts of collision detection algorithms.

The second approach, which could be described as the "discrete view", involves the geometric analysis of bodies at a fixed instant of time. The goal under this approach is to produce algorithms that solve a single, static instance of a problem with optimal asymptotic time complexity. Work of this sort includes an approximately $O(n \log n)$ algorithm by Cameron and Culley[5] for computing the distance between two convex polyhedra with n vertices. Gilbert, Johnson, and Keerthi[19] describe an algorithm for computing the minimum distance between convex polyhedra with n vertices; the algorithm appears to have a running time somewhat larger than $O(n)$ but smaller than $O(n \log n)$. Related work by Gilbert and Foo[17] extends the algorithm to handle smooth convex shapes and concludes

that smooth representations become more efficient than polyhedral representations in the neighborhood of $n = 100$. For the problem of determining disjointness of two convex polyhedra with n vertices (without regard to the distance between them), an $O(n)$ algorithm is readily available, based on Megiddo's work on linear programming problems with constant dimension[37,43].

Unfortunately, the above algorithms, although they achieve small asymptotic time complexity, do so at the price of large run time constants. For example, the linear time algorithm for deciding disjointness between convex polyhedra with n vertices is not practical unless n is quite large. More importantly, the use of discrete view algorithms essentially ignores any similarity that may exist with the current collision/contact determination problem and previous collision/contact determination problems. Proper use of previous information can result in very simple, yet efficient collision/contact determination algorithms; such algorithms are presented in Chapter 5. The collision detection algorithm in Gilbert, Johnson, and Keerthi[19] is structured so that it can use information previously computed to obtain a faster running time, but this is not a main consideration of the algorithm. Likewise, work by Cundall[10] on dynamic simulation of polyhedral blocks uses a collision/contact determination algorithm that specifically exploits information obtained from the previous determination. However, Cundall's approach is somewhat more complicated than needed for dynamic simulation and is restricted to polyhedra. The philosophy of collision/contact determination presented in Chapter 5 was largely inspired by Cundall's approach.

4.2 Impulse computation

The computation of impulses between two bodies that collide at a single contact point without friction is straightforward, as it involves the solution of a single linear

equation with one unknown. If equality constraints involving one of the colliding bodies exist (as happens when an articulated rigid body undergoes a collision), additional impulsive constraint forces are needed to maintain the equality constraint. In this case, the impulses must be computed simultaneously, so a larger set of linear equations must be solved; this is simply a special case of solving equality constrained dynamics problems, and is not particularly difficult.

If the contact involves friction, analysis of the problem is more complicated, but the actual computation itself is simple, for two-dimensional systems. Wang and Mason[34,54] discuss the case of collisions with friction at a single contact point in two dimensions, and give a complete treatment of the problem. Their analysis of the problem yields simple formulas for the impulses arising from a collision. However, Wang and Mason's results are difficult to extend to three dimensions, which can be seen by considering Keller[24], who presents a differential description of the three-dimensional motion of bodies undergoing a collision with friction at a single contact point. The differential equations appear easy to solve numerically, but very difficult to treat analytically. Keller's analysis involves regarding the collision as occurring over an arbitrarily small time interval, and examining the limiting behavior of the system as the time interval is brought to zero. We shall employ the same sort of analysis in considering impulses with friction in chapters 12 and 13. An animation system described by Moore and Wilhelms[39] uses penalty methods to compute constraint forces, but computes constraint impulses directly for collisions involving a single contact point. Moore and Wilhelms use a simplified description of the Coulomb friction law to compute frictional impulses at a single contact point in three dimensions. We will use a similar simplification, applied to a number of contact points simultaneously (see below).

Collisions can also be regarded as occurring simultaneously at a number of contact points. For example, if a cube is dropped onto a level plane so that all

four vertices of the bottom face of the cube strike the plane together, the collisions between the vertices and the plane can be modeled as occurring at discrete instants of time, by the above methods, or as occurring simultaneously. There are large computational savings to be gained by modeling the collisions as occurring simultaneously. Cremer[8] discusses the advantages and disadvantages of the simultaneous model of collisions, and Featherstone[13] shows how to formulate the necessary equations for the frictionless case. The impact model for simultaneous collisions used in this thesis agrees with Featherstone's formulation, for the frictionless case. Lötstedt[30] computes simultaneous frictional impulses in three dimensions by using a modification of the Coulomb friction law that causes impacts to dissipate as much energy as possible. In general, it is unclear how to deal correctly with simultaneous impacts with friction, in either two or three dimensions. The simulator described in this thesis computes simultaneous frictional impulses for three-dimensional bodies, based on a modification of the model proposed by Lötstedt. As stated, for frictionless systems, the model described in this thesis is the same as Featherstone's. However, for systems with friction, the model does not always properly conserve energy for impacts[3,54]. The general problem of correctly computing frictional impulses in three dimensions, and simultaneous frictional impulses in either two or three dimensions is not addressed by this thesis.

4.3 Penalty methods for constraint forces

A vast number of simulations[10,22,26,36,38,39,50,51,55] have employed the penalty method as described in Section 2.4 to enforce non-penetration constraints; applications include the simulation of deformable bodies, cloth, and articulated rigid bodies. The penalty method is a very attractive model in some respects; it is extremely simple to implement, and very versatile. As a numerical method, it is

nowhere near as complex as the methods we will consider throughout most of this thesis. In particular, exploiting parallelism and/or hardware implementations with the penalty method is much easier to imagine than with the methods we will be discussing throughout most of this thesis. Practically speaking though, the use of the penalty method in essentially dynamic conditions leads to stiff differential equations and is not a suitable method for the type of simulation we are interested in. The penalty method is reasonable only for situations that are essentially static; that is, where there is little relative motion between bodies and little to no change in constraint forces between bodies. We will consider the additional question of the theoretical correctness of the penalty method as a simulation algorithm in Chapter 7.

4.4 Exact methods for constraint forces

In comparison with the penalty method, relatively few dynamics systems use exact methods to calculate constraint forces. Descriptions of the system of equations in terms of a quadratic program for calculating multiple constraint forces without friction (for polyhedra) appear in Kilmister and Reeve[25] and Ingleton[23]. Cottle[7] clarifies and simplifies some of Ingleton's results. Kilmister and Reeve, Ingleton, and Cottles' work will be our basic starting point when we consider systems with multiple contact points in Chapter 8. We will make use of the same quadratic programming formulation and extend the contact geometry allowed to include smooth curved surfaces.

All of the above work took place before the discovery of the importance of P and NP time-complexities. As a result, the computational complexity of computing multiple contact forces was not of concern in any of the above work. In work done subsequent to this discovery, Erdmann[12] discusses the problem of computing constraint forces without friction, and notes that the problem is simply solved by

an exhaustive search method, requiring exponential time. The earliest description of a simulator using exact methods to calculate constraint forces (by quadratic programming) appears to be by Lötstedt[29]. Lötstedt notes that the quadratic program can be solved efficiently because it is convex. More recently, Featherstone[13] has described a heuristic algorithm for solving the quadratic programs generated for configurations without friction that is intended to improve upon the obvious exhaustive search method. However, Featherstone's heuristic requires that configurations of bodies be statically determinate; that is, the normal forces that prevent inter-penetration for the configuration must be unique. This is a very restrictive condition though, which we will not want to impose on our simulations. In this thesis, the computational complexity of computing multiple contact forces will be of great interest to us.

4.5 Friction

Adding a model of friction to simulations using the penalty method is straightforward. Computing both friction and constraint forces using exact methods is considerably more difficult. For two-dimensional problems, an obvious exhaustive search method exists, but is not suitable to simulations with many contact points because it requires exponential time in the number of contact points. Mason and Wang[35] discuss the computation of friction and constraint forces in systems with only a single contact point (where the time complexity of the computation is not an issue). Lötstedt[28], Erdmann[12], and Mason and Wang all discuss interesting properties of the Coulomb friction model in relation to rigid body simulation. Their discussions and examples of the unexpected consequences of introducing Coulomb friction provide the starting point for much of the discussion on friction in this thesis. In particular, Erdmann describes the problem of computing contact forces

in terms of configuration space (Chapter 6). We will borrow from Erdmann by using configuration space to develop the dynamics of systems with a single point of contact, although we will dispense with configuration space when we move on to systems with multiple contact points and friction, as the configuration space approach offers no additional insights into the problem. Additionally, Lötstedt[30] proposes a modification of the Coulomb friction law for rigid body simulation in conjunction with a practical computational solution method for the modified friction model. We will examine Lötstedt's modification and subsequent solution method in Chapter 14. Again, the computational complexity in computing multiple contact forces with friction will be of considerable interest.

Chapter 5

Collision/Contact Determination

In summarizing previous work on collision/contact determination in Section 4.1, we noted that of the two groupings of algorithms (the discrete view and the continuum view), the continuum view, which is commonly used in robotics, is not applicable to the dynamics problem we are interested in. The other approach, the discrete view, could be used to solve the collision/contact determination problem, but at the price of ignoring any knowledge due to the inherent spatial continuity of the problem as a function of time. Instead, a third approach, which we will call the "coherence view", will be adopted and used to solve the collision/contact determination problem. The use of coherence need not be confined to collision/contact determination; in fact, critical use will be made of coherence in computing constraint and friction forces. In general, methods that exploit coherence should be considered throughout the entire simulation process.

As stated in Chapter 3, the simulation process consists of the repeated computation of $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t)$ at various times t. The numerical ODE solver is responsible for choosing the values of t at which the state derivative is to be computed. For any reasonably complicated simulation, the values of t chosen are such that the state

does not change greatly between successive values of t. As a result, there is almost always great geometric coherence between successive time steps. It is difficult to formally quantize the degree of coherence, and we will not attempt to do so. In practice, the degree of coherence present during simulations has been such that the algorithms presented in this chapter are much faster than their discrete view counterparts. Additionally, their simplicity has greatly decreased the programming difficulties that arise with any complex simulation system.

The coherence view attempts to capitalize on geometric coherence to achieve both simpler and more efficient collision/contact determination algorithms.[1] The series of collision/contact determination problems that need to be solved during the simulation are considered, according to the coherence view, as separate problems; however, the problems are considered to be related, and wherever possible, information from the solution of the previous problem is used to aid in the solution of the current problem. Using this view, algorithms that are both faster *and* simpler than their discrete view algorithms counterparts are possible.

In this chapter, we will discuss three separate problems of collision/contact determination. The first problem we consider is pairwise collision/contact determination between convex polyhedra. Next, pairwise determination between convex curved surfaces or a convex curved surface and a convex polyhedron is considered. Last, we will consider a simple coherence-based algorithm for detecting all pairwise-intersections among a group of rectangular solids aligned with the coordinate axes. This algorithm is used at the beginning of the collision/contact determination to quickly cull out pairs of objects whose rectangular bounding boxes do not overlap;

[1] For those interested in building a practical, robust simulation system with a rich variety of object geometries, it is sometimes hard to say which is more valuable—increased simplicity or decreased running time. Given the tendency complicated geometrical algorithms have to get stuck and abort on the special cases that always arise in practice, it sometimes seems that slow but simple algorithms end up saving time over a complicated algorithm that requires the simulation to be run and re-run and re-re-run and ...

such a step is necessary to avoid performing $O(n^2)$ pairwise collision/contact determinations among a group of n bodies. Using the algorithms developed to solve these three problems, we will be able to perform collision/contact determination on any body that can be described as the union of convex polyhedra and strictly convex closed surfaces. These polyhedra and curved surfaces will be referred to as *primitives*. For curved surfaces primitives, only closed surfaces with strictly positive curvature at every point will be allowed for collision/contact determination. Note that although polyhedral primitives must also be convex, this is not really a restriction, since any polyhedron can be decomposed into a union of convex polyhedra. The general case of collision/contact determination between non-convex curved surfaces is beyond the scope of this thesis; however, this limitation will *not* be imposed when considering the dynamics of bodies in subsequent chapters.

The discussion of the collision/contact determination step in Section 3.2 made mention of the fact that the collision/contact determination step was responsible for estimating the time of a collision. This will be covered in Chapter 6.

5.1 Convex polyhedra

Our primary mechanism for exploiting coherence will be through the use of *witnesses* to the decision problem of inter-penetration between primitives. A witness is some piece of information that can be used to quickly answer a decision problem. We will utilize coherence by caching witnesses from one time step to the next; hopefully a witness from the previous time step will be a witness during the current time step.

Since primitives are convex, a pair of primitives do not inter-penetrate if and only if a separating plane between them exists. A separating plane between two objects is a plane such that each object lies in a different half-space of the plane. A given plane can be verified to be a separating plane between two polyhedra in time $O(n)$

where n is the total number of vertices in the two polyhedra. Thus, a separating plane is a witness to the fact that two convex polyhedra do not inter-penetrate.

We would like to structure our algorithms so that their running time decreases as coherence increases. The basic philosophy behind utilizing coherence through cached witnesses is to choose a very simple, fast strategy that proposes a new witness based on the cached witness, and then quickly attempts to verify the new witness. If the proposed witness is invalid, a second, more time-consuming step is executed to find a valid witness. The idea is that the first step is fast enough, and succeeds often enough to more than make up the time lost on the occasions when it does fail.[2] In particular, the strategy should be extremely simple, so that it has both a small asymptotic time complexity and run time constant. The cost of initially finding a witness (for the very first time step of the simulation, or the first time two bodies become close enough to require more than a bounding box test) is unavoidable; either exhaustive search algorithms or the discrete view algorithms described in Section 4.1 can be used to initially compute these witnesses.

As mentioned in Section 4.1, algorithms by Cundall[10] and Gilbert, Johnson and Keerthi[19] come close to this philosophy. In particular, the algorithm Gilbert, Johnson and Keerthi describe could be initialized with the solution of the previous problem to decrease the running time; however, the initialized algorithm is still much more complicated than is needed for situations with a high degree of coherence. Cundall's approach to collision/contact detection is much simpler, and in fact inspired the coherence philosophy just presented. Cundall called a separating plane between two convex polyhedra a *common plane*. For each pair of convex polyhedra requiring consideration, he used numerical techniques to initially determine a separating plane that was approximately equidistant from each polyhedron of the pair; if such a plane

[2] In general, such a philosophy can be implemented hierarchically, as is done for memory caching on computers, but we will not bother to do so.

could not be found, then clearly the two polyhedra had inter-penetrated. He then noted several important facts about the separating plane. If the polyhedra were disjoint, this was easily shown because each polyhedra could trivially be shown to lie some distance from the separating plane. Furthermore, if the polyhedra were contacting, then the contact points were easy to find because they would have to lie on a common plane of contact (hence the term *common plane*); this greatly reduced the complexity of determining the contact points. Most important was Cundall's realization that it was not necessary to compute separating planes from scratch at each time step; a simple numerical iteration could compute a new separating plane by using the old separating plane as a starting point, because of the geometric coherence between successive time steps.

Cundall's approach is good, but it can be made even better. If a pair of convex polyhedra are disjoint or contacting (but not inter-penetrating), then a separating plane exists with the following property: either the plane embeds a face of one of the polyhedra, or the plane embeds an edge of one of the polyhedra and is parallel to an edge of the other polyhedra. We will call the face or edges in question the *defining* face or edges. Instead of caching a specific plane in $\mathbf{R}^3$ as the witness and numerically updating it, the defining face or edges are cached as the witness. At the next time step, a separating plane is formed from the defining face or edges, and then verified. The verification is simple; the vertices of the two polyhedra are checked to see that they lie on opposite sides of the separating plane. Since this verification takes $O(n)$ time (where n is the number of vertices), a linear time algorithm results whenever the cached face or two edges form a separating plane. Clearly, in addition to having a linear time cost, this algorithm will be considerably faster than other $O(n)$ algorithms, such as the linear programming algorithm discussed in Section 4.1.

However, an additional improvement can be made so that this verification can usually be performed in sublinear time. It is not always necessary to test all the

vertices of a polyhedron to verify that the polyhedron lies completely on one side of the separating plane. Instead, for each polyhedron, the vertex lying closest to the separating plane is cached. If at the next time step that cached vertex is closer to the separating plane than all of its neighbors, then by convexity all of the vertices of the polyhedron must lie on the same side of the separating plane. This gives the result that disjointness or contact between convex polyhedra can usually be determined in sublinear time when coherence is high.

On those occasions when the cached face or two edges fails to form a valid separating plane, faces or edges adjacent to the previously cached face or edges can be used to form a possible separating plane; however, this happens infrequently enough that it may be simpler to start from scratch and compute a new separating plane without using any prior knowledge. It should be obvious that the linear time cost of verifying a separating plane is of a much different order than the linear time cost of using a technique like linear programming[43] to find a separating plane; although both algorithms have $O(n)$ time complexity, their run time constants are considerably different.

Once the separating place has been found, the contact region between the two polyhedra is determined, assuming the polyhedra are not disjoint. Contact points between the two polyhedra can only occur on the separating plane. Given the separating plane, the contact points can be quickly and efficiently determined by comparing only those faces, edges, and vertices of the polyhedra that are coincident with the separating plane.

However, if no separating plane can be found, then the two polyhedra must be inter-penetrating. When two polyhedra inter-penetrate, it is almost always the case that either a vertex of one polyhedron is inside the other, or an edge

of one polyhedron has intersected a face of the other.[3] In this case, the inter-penetrating vertex, or intersecting edge and face are cached as a witness to the inter-penetration. Since this indicates a collision at some earlier time, the simulator will back up and attempt to compute $\frac{\mathrm{d}}{\mathrm{dt}}\mathbf{Y}(t)$ at some earlier time. Until the collision time is determined, the first action taken by the collision/contact determination step will be to check the cached vertex or edge and face to see if they indicate inter-penetration. Thus, until the collision time is found, states in which the inter-penetration still exists are identified as such with a minimum of computational overhead. In Chapter 6, we will show how the collision time may be estimated once an inter-penetration is detected.

5.2 Convex curved surfaces

The surfaces considered in this thesis are assumed to be twice-differentiable and without boundary; in this chapter, surfaces are also assumed to be strictly convex. Before we can consider collision/contact determination between curved surfaces, we must decide how the curved surfaces are to be represented.

There are basically two choices—surfaces can either be described by implicit functions or parametric functions. In the implicit case, a surface is described as a time-varying function $F(p,t)$ where p is a point in world space. At time t, point p is on the surface represented by F if

$$F(p,t) = 0. \tag{5–1}$$

In the parametric case, a surface is described by a time-varying function $S(u,v,t)$ where u and v are scalars and $S(u,v,t)$ is a point on the surface. If F and S

[3] An exception is the following. Stack two cubes of equal size atop one another so that their contacting faces exactly coincide. Lower the top one. This produces an inter-penetration such that no vertex is inside either cube, and no edge penetrates through any face.

describe the same surface, then

$$F(S(u, v, t), t)$$

is identically zero.

We will not attempt any formal manipulations of F and S; that is, it is assumed F and S are given to us as "black-box" functions. The only allowable operations on F or S are evaluating F or S, or their first or second derivatives at specific values of p and t (for F) and u, v and t (for S). For simplicity, derivations will be developed in the text only for surfaces described implicitly. Matching derivations for parametrically defined surfaces are found in Appendix A.

We will consider two convex curved surfaces A and B that are described implicitly by time-varying functions $F(p, t)$ and $G(p, t)$. Any point p_0 lying on A at time t satisfies $F(p_0, t) = 0$. Furthermore, a point p_1 that is *inside* of A satisfies $F(p_1, t) < 0$, while a point p_2 that is *outside* of A satisfies $F(p_2, t) > 0$. The other surface B is similarly defined in terms of G, and we will sometimes refer to F and G as both functions and surfaces.

The surface normal of F at point p and time t is written as $\nabla F(p, t)$, and is a column vector. Formally,

$$\frac{\partial}{\partial p} F(p, t) = F'(p, t) = \nabla F(p, t)^T.$$

Since F is negative on the interior of A and positive on the exterior of A, the surface normal $\nabla F(p, t)$ is always outwards pointing.

5.2.1 Extremal points of curved surfaces

First, let us consider the problem of verifying that two convex curved surfaces A and B do not inter-penetrate. Since A and B are strictly convex smooth surfaces,

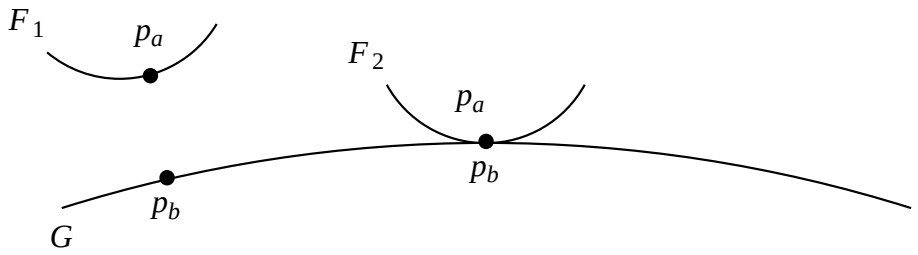

Figure 5.1: The minimum distance points p_a and p_b between two convex surfaces.

if they do not inter-penetrate then there is a unique pair of points p_a and p_b lying on A and B that minimize the distance between A and B (Figure 5.1). If these points can be found, then they clearly demonstrate that A and B are not inter-penetrating. Furthermore, if the bodies are in contact, then $p_a = p_b$ is the contact point between A and B. Thus, in both cases, the pair of points p_a and p_b minimizing the distance between A and B seems a good choice for a witness.

What about the case when A and B inter-penetrate? In this case, the minimum distance between A and B is zero. In detecting contact between surfaces, if the depth δ of inter-penetration between A and B falls below some numerical threshold ϵ, then A and B must be considered to be contacting, rather than inter-penetrating. For instance, in Figure 5.2, suppose the inter-penetration depth between F_1 and G is δ_1; similarly, let δ_2 be the inter-penetration depth between F_2 and G. From the figure, $\delta_1 < \delta_2$. If

$$\delta_1 < \epsilon < \delta_2$$

then F_1 and G would be considered to be contacting, while F_2 and G would be considered to be inter-penetrating. Thus, when surfaces are not disjoint, the minimum distance, which is always zero, is not a useful piece of information. Although the points of intersection are not directly useful, one thought might be to examine the angle between surfaces at intersection points; however, as Figure 5.2

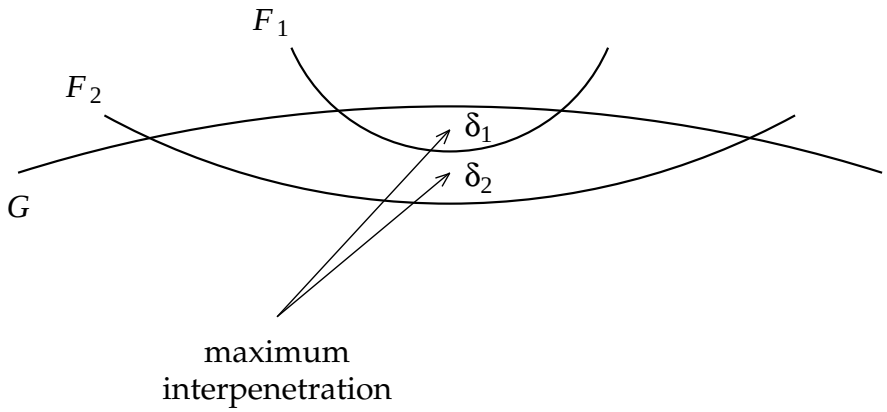

Figure 5.2: Inter-penetration between surfaces. Surface F_1 intersects G at a sharper angle than surface F_2 does, yet the maximum inter-penetration depth δ_2 between F_2 and G is larger than the maximum inter-penetration depth δ_1 between F_1 and G.

shows, the angle of intersection is independent of the depth of inter-penetration. For inter-penetration, the points of interest are the points that indicate the *maximum* distance between A and B in their intersection. Thus, a suitable witness for inter-penetration are the points of maximum distance between A and B.

We can formalize and unify our definition of a witness for both the inter-penetrating situation and the non-inter-penetrating situation by defining the general concepts of the *extreme distance* between A and B, and the *extremal points* between A and B. The extreme distance between A and B is defined as follows. If A and B are disjoint, then the extreme distance between A and B is just the normal minimum distance between A and B. If A and B are in contact, then the extreme distance is zero. If A and B have inter-penetrated, then the extreme distance is the maximum distance between A and B in their intersection. The extremal points of A and B are the two points p_a and p_b, lying on A and B respectively, that realize the extremal distance (Figure 5.3).

The extremal points can be computed by considering the following four necessary

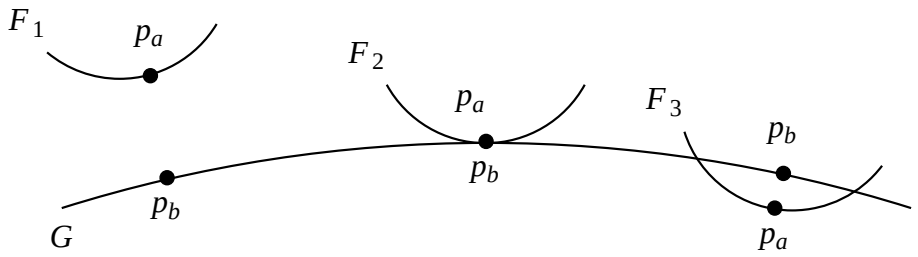

Figure 5.3: The definition of the extremal points. The extremal points p_a and p_b between F_1 and G are the points of minimum distance. The same is true for F_2 and G. The extremal points p_a and p_b between F_3 and G are the points that maximize the distance between F_3 and G in their intersection.

conditions for a pair of points p_a and p_b to be extremal at time t:

$$\begin{cases} E_1: & \nabla F(p_a,t) + \lambda_2 \nabla G(p_b,t) = \mathbf{0} \\ E_2: & F(p_a,t) = 0 \\ E_3: & G(p_b,t) = 0 \\ E_4: & (p_b - p_a) + \lambda_1 \nabla G(p_b,t) = \mathbf{0}. \end{cases} \tag{5–2}$$

The variables λ_1 and λ_2 are unconstrained scalar values (with no relation to the λ of Section 2.5) and $\mathbf{0}$ is a column vector of zeroes. Condition E_1 guarantees that the surface normals of F and G at p_a and p_b are colinear. Conditions E_2 and E_3 guarantee that p_a and p_b are points on A and B, and condition E_4 guarantees that p_b's displacement from p_a is colinear to the surface normals. Figure 5.4 shows the geometric intuition behind these four conditions; they are derived based on the Lagrange multiplier formulation for constrained minimization (hence the choice of λ for the multipliers of ∇G).

These four conditions form a non-linear set of eight equations in eight unknowns. However, the conditions are necessary but not sufficient for p_a and p_b to be extremal points. Because Equation (5–2) is non-linear, it will have multiple solutions. For example, the points that globally *maximize* the distance between A and B always

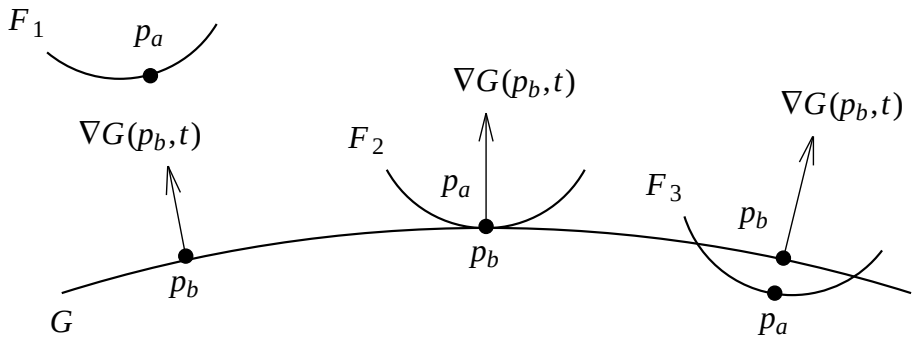

Figure 5.4: Geometric conditions for the extremal points. In order for p_a and p_b to be extremal, the surface normals ∇F and ∇G must be parallel, and point directly from p_a to p_b.

satisfy Equation (5–2), but are not a solution we are interested in. Non-linear equations are usually solved numerically using iterative methods. The numerical method proceeds from some initial estimate of the solution to an exact solution (within numerical tolerances). The initial estimate determines which solution is computed[11].

If we are initially computing the extremal points, we need an initial estimate sufficiently close to the extremal points to avoid computing a spurious solution. For implicitly defined convex surfaces, the following simple device can be used. A line segment passing though the centroids of A and B is constructed. The intersection of this line segment and A is roughly computed; that is, a point p on the line segment satisfying $F(p,t) = 0$ is computed. This can be done very easily using bisection, and since only a rough approximation is needed, a small number of bisection steps will be sufficient (ten or less). Similarly, an intersection between the line segment and B is computed. Using the two intersections as initial estimates for p_a and p_b, Equation (5–2) is iteratively solved to find p_a and p_b exactly. For parametric surfaces, a coarse mesh of surface points can be generated and stored for each surface. The parametric coordinates of the minimum distance pair of mesh points

can be used as an initial estimate. The initial solution of Equation (5–2) requires a fair amount of work, in terms of generating an initial estimate of p_a and p_b, and then solving Equation (5–2).

However, once the extremal points p_a and p_b are determined, they are cached for the next step. In subsequent time steps, the cached values of p_a and p_b are used as initial estimates in the numerical solution of Equation (5–2). Because of coherence, the cached values p_a and p_b will greatly reduce the number of iterative steps needed to numerically solve Equation (5–2). Furthermore, the accuracy of the initial estimate can be substantially improved if the *derivatives* of the extremal points are cached at each time step. Suppose that at time t_0, the extremal points p_a and p_b, and their derivatives $\dot{p}_a$ and $\dot{p}_b$ are cached. If the next time step takes place at time $t_0 + \Delta t$, p_a and p_b at that time can be estimated as

$$p_a + \Delta t\, \dot{p}_a \qquad \text{and} \qquad p_b + \Delta t\, \dot{p}_b \tag{5–3}$$

respectively. Improving the accuracy of the initial estimate of p_a and p_b increases the overall speed of the algorithm. Sections 6.3 and 6.4 will discuss the computation of $\dot{p}_a$ and $\dot{p}_b$.

5.2.2 Polyhedron/curved surface determination

Essentially the same method for collision/contact determination between a pair of curved surfaces can be used for collision/contact determination between a curved surface and a polyhedron. The concept of extremal points is again used, however, different formulations are needed depending on whether the extremal point on the polygon lies on a face, an edge, or a vertex.

Let the curved surface be denoted A, and be described implicitly by a function F as before. Let B denote the polyhedron. Let the plane equation of a face of B be implicitly described by a function $G(p, t)$. Necessary conditions for the

extremal points to occur between this face and A are given by Equation (5–2), with the restriction that p_b actually lie inside the face. A simple initial step to compute p_a and p_b is to test each face of B against A, stopping if valid extremal points are found. Clearly, not all faces of B need be considered; faces of B whose outwards surface normal points away from A can obviously be ignored. However, if no extremal points exist between a face of B and A, then the edges of B must be tested.

An edge of B with endpoints P_0 and P_1 can be parameterized as the set of points $P_0 + c\,r$, where $r = P_1 - P_0$ and $0 \leq c \leq 1$. Necessary conditions for p_a and $p_b = P_0 + c\,r$ to be extremal are

$$\begin{cases} \nabla F(p_a, t)^T r = 0 \\ F(p_a, t) = 0 \\ (p_a - (P_0 + c\,r)) + \lambda_1 \nabla F(p_a, t) = \mathbf{0} \end{cases} \tag{5–4}$$

along with the condition that p_b be contained in the edge; that is, $0 \leq c \leq 1$. If no extremal points exist between an edge of B and A, then an extremal point will exist between a vertex P_0 of B and A. The conditions for a vertex P_0 of B and p_a to be extremal are

$$\begin{cases} F(p_a, t) = 0 \\ (p_a - P_0) + \lambda_1 \nabla F(p_a, t) = \mathbf{0}. \end{cases} \tag{5–5}$$

Once an extremal point is found between a face, edge, or vertex, the extremal points are cached, along with the face, edge or vertex on which the extremal point occurred. Subsequent solutions of the extremal points then take advantage of knowing which face, edge, or vertex to test to find the extremal points, along with a good starting estimate of the extremal points. In all cases, the velocity $\dot{p}_a$ is cached. If the extremal point lies on a face, $\dot{p}_b$ is also cached, and if the extremal point lies on an edge, $\dot{c}$ is also cached.

5.3 Bounding boxes

To reduce the number of pairwise collision/contact determinations necessary, a bounding box hierarchy can be imposed on the objects in the simulation environment. If two bounding boxes are found not to overlap, no further comparisons involving the contents of the boxes are needed. Given a collection of n rectangular bounding boxes, aligned with the coordinate axes, we would like to efficiently determine all pairs of boxes that overlap. A naive pairwise comparison of all pairs requires $O(n^2)$ work, and can clearly be improved. Discrete view algorithms for this problem in $\mathbf{R}^3$ exist with a time complexity of $O(nlogn + k)$ where k is the number of pairwise overlaps; a general result is that the problem can be solved in time $O(nlog^{d-2}n + k)$ for d-dimensional bounding boxes[43]. Again, a coherence view algorithm exists with substantially better performance.

5.3.1 The one-dimensional case

Consider the problem of detecting overlap between *one*-dimensional bounding boxes, aligned with the coordinate system. Such a bounding box can be described simply as an interval $[b, e]$. Let us consider a list of n such intervals, with the ith interval being $[b_i, e_i] \subset \mathbf{R}^1$. The problem is then defined to be the determination of all pairs i and j such that the intervals $[b_i, e_i]$ and $[b_j, e_j]$ intersect.

The problem can be solved initially by a *sort and sweep* algorithm. A sorted list of all the b_i and e_i values is created, from lowest to highest. The list is then swept, and a list of *active* intervals, initially empty, is maintained. Whenever some value b_i is encountered, all intervals on the active list are output as overlapping with interval i, and interval i is then added to the list (Figure 5.5a). Whenever some value e_i is encountered , interval i is removed from the active list (Figure 5.5b). The cost of this process is $O(n \log n)$ to create the sorted list, $O(n)$ to sweep through the list,

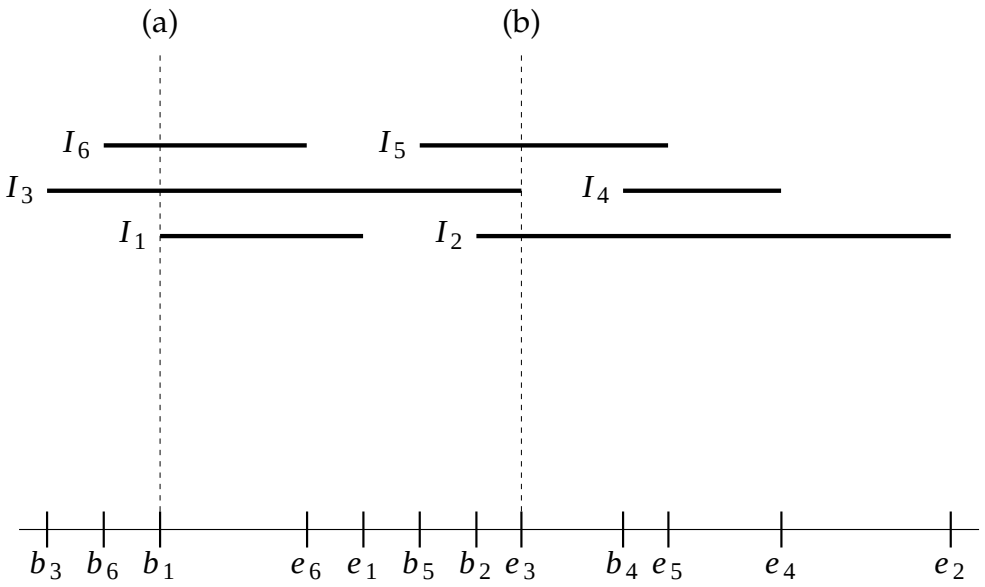

Figure 5.5: The sweep/sort algorithm. (a) When b_1 is encountered, the active list contains intervals 3 and 6; interval 1 is reported to overlap with these two intervals. Interval 1 is added to the active list and the algorithm continues. (b) When e_3 is encountered, the active list contains intervals 2, 3 and 5. Interval 3 is removed from the active list.

and $O(k)$ to output each overlap. This gives a total cost of $O(n \log n + k)$, and is an optimal algorithm for initially solving the problem.

Subsequent comparisons can be improved as follows. First, there is no need to use an $O(n \log n)$ algorithm to form the sorted list of b_i and e_i values. It is considerably more efficient to start with the order found for b_i and e_i values from the previous time step; if coherence is high, this ordering will be nearly correct. A sorting method called an *insertion sort*[46] is used to permute the "nearly sorted" list into a sorted list. The insertion sort algorithm works by moving items towards the beginning of the list, until a smaller item is encountered. Thus, the second item is interchanged with the first if necessary, then the third item is moved towards the beginning of the list until its proper place is found, and so on; each movement of an item indicates a change in the ordering of two values. After the last item on the list

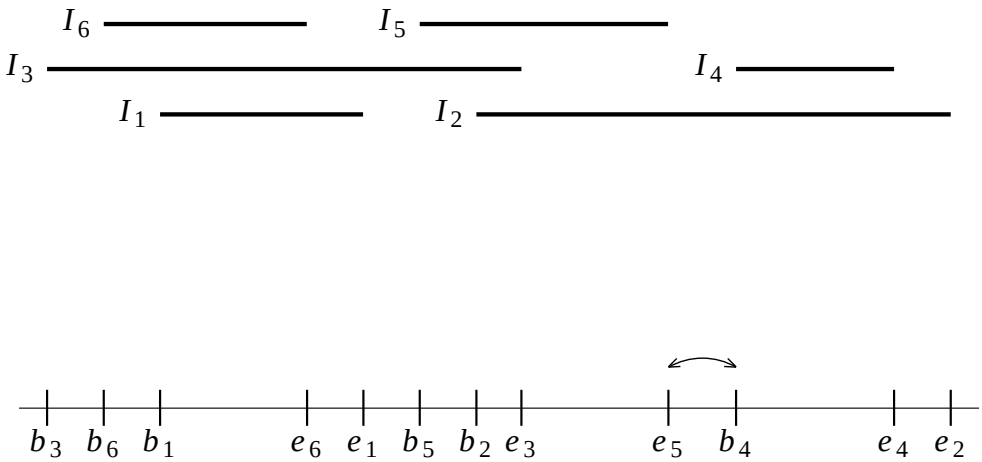

Figure 5.6: A coherence-based method of detecting overlaps. The order produced in Figure 5.5 is nearly correct for this arrangement of intervals. Only b_4 and e_5 need to be exchanged. When the exchange occurs, the change in overlap status between interval 4 and 5 is detected.

has been processed, the list is sorted. Such a sort takes time $O(n+s)$ where s is the number of "swaps" necessary to reorder the list. For example, the only difference between Figures 5.6 and 5.5 is that interval 4 has moved to the right. Starting from the ordered list of b_i and e_i values of Figure 5.5, only a single exchange is necessary to sort the list for Figure 5.6. The insertion sort is not recommended as a sorting procedure in general, since it may require $O(n^2)$ exchanges; however, it is a good algorithm for sorting a nearly sorted list, which is what occurs in our highly coherent environment. To complete the algorithm, note that if two intervals i and j overlap at the previous time step, but not at the current time step, one or more exchanges involving either a b_i or e_i value and a b_j or e_j value must occur. The converse is true as well when intervals i and j change from not overlapping at the previous time step to overlapping at the current time step.

Thus, if we maintain a table of overlapping intervals at each time step, the table can be updated at each time step with a total cost of $O(n+s)$. An optimal algorithm

would have time complexity $O(n+k)$, so our algorithm, with its $O(n+s)$ complexity is efficient when $s-k$ is small. Assuming coherence, the number of exchanges s necessary will be close to the actual number k of changes in overlap status, and the extra $O(s-k)$ work will be negligible. Thus, for the one-dimensional bounding box problem, the coherence view yields an efficient algorithm of extreme (if not maximal) simplicity that approaches optimality as coherence increases.

5.3.2 The three-dimensional case

Efficient discrete view algorithms for solving the bounding box intersection problem in $\mathbf{R}^3$ are much more complicated than the sweep and sort method for the one-dimensional case. However, these algorithms all have in common a step that is essentially a sort along a coordinate axis, as in the one-dimensional case. Each bounding box is described as three independent intervals $[b_i^{(x)}, e_i^{(x)}]$, $[b_i^{(y)}, e_i^{(y)}]$, and $[b_i^{(z)}, e_i^{(z)}]$ which represent the intervals spanned on the three coordinate axes by the ith bounding box. Thus, our first thought towards improving the efficiency of a discrete view algorithm for coherent situations would be to sort a list containing the $b_i^{(x)}$ and $e_i^{(x)}$ values, and similarly for the y and z axes. Again, such a step will involve $O(n+s)$ work, where now s is the total number of swaps involved in sorting all three lists. However, if we observe that checking two bounding boxes for overlap is a constant time operation, it follows that if we simply check bounding boxes i and j for overlap whenever an exchange is made between values indexed by i and j (on any coordinate axis), we will detect all changes in overlap status in $O(n+s)$ time.

Again, we can maintain a table of overlapping bounding boxes, and update it at each time step in $O(n+s)$ time. The extra work involved is again $O(s-k)$. For the three-dimensional case, extra work can occur if the extents of two bounding boxes

change on one coordinate axis without an actual change of their overlap status. In practice, the extra work done has been found to be completely negligible, and the algorithm runs essentially in time $O(n + k)$.