Fast Contact Force Computation for Nonpenetrating Rigid Bodies

David Baraff

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

A new algorithm for computing contact forces between solid objects with friction is presented. The algorithm allows a mix of contact points with static and dynamic friction. In contrast to previous approaches, the problem of computing contact forces is not transformed into an optimization problem. Because of this, the need for sophisticated optimization software packages is eliminated. For both systems with and without friction, the algorithm has proven to be considerably faster, simpler, and more reliable than previous approaches to the problem. In particular, implementation of the algorithm by nonspecialists in numerical programming is quite feasible.

1. Introduction

In recent work, we have established the viability of using analytical methods to simulate rigid body motion with contact[1,2,3]. In situations involving only bilateral constraints (commonly referred to as "equality constraints"), analytical methods require solving systems of simultaneous linear equations. Bilateral constraints typically arise in representing idealized geometric connections such as universal joints, point-to-surface constraints etc. For systems with contact, unilateral (or "inequality") constraints are required to prevent adjoining bodies from interpenetrating. In turn, the simultaenous linear equations arising from a system of only bilateral constraints must be augmented to reflect the unilateral constraints; the result is in general an inequality-constrained nonlinear minimization problem.

However, analytical techniques for systems with contact have yet to really catch on in the graphics/simulation community. We believe that this is because of the perceived practical and theoretical complexities of using analytical techniques in systems with contact. This paper has two goals, one of which is to address these concerns: in particular, we present analytical methods for systems with contact that can be practically implemented by those of us (such as the author) who *are not specialists in numerical analysis or optimization.* These methods are simpler, reliable, and faster than previous methods used for either systems with friction, or systems without friction.

Our other goal is to extend and improve previous algorithms for computing contact forces with friction[3]. We present a simple, fast algorithm for computing contact forces with friction. The restriction of our algorithm to the frictionless case is equivalent to an algorithm described in Cottle and Dantzig[4] (but attributed to Dantzig) for solving linear complementarity problems. It is not our intention to reinvent the wheel; however, it is necessary to first understand Dantzig's algorithm and why it works for our frictionless sytems before going on to consider the more general solution algorithm we propose to deal with friction. We give a physical motivation for Dantzig's algorithm and discuss its properties and implementation in section 4. For frictionless systems, our implementation of Dantzig's algorithm compares very favorably with the use of large-scale, sophisticated numerical optimization packages cited by previous systems[11,7,8,6]. In particular, for a system with n unilateral constraints, our implementation tends to require approximately three times the work required to solve a square linear system of size n using Gaussian elimination. Most importantly, Dantzig's algorithm, and our extensions to it for systems with friction, are sufficiently simple that nonspecialists in numerical programming can implement them on their own; this is most assuredly *not* true of the previously cited large-scale optimization packages.

Interactive systems with bilateral constraints are common, and there is no reason that moderately complicated interactive simulation with collision and contact cannot be achieved as well. We strongly believe that using our algorithms, interactive simulations with contact and friction are practical. We support this claim by demonstrating the first known system for interactive simulations involving contact and a correct model of Coulomb friction.

2. Background and Motivation

Lötstedt[10] represents the first attempt to compute friction forces in an analytical setting, by using quadratic programming to compute friction forces based on a simplification of the Coulomb friction model. Baraff[3] also proposed analytical methods for dealing with friction forces and presents algorithms that deal with dynamic friction (also known as sliding friction) and static friction (also known as dry friction). The results for dynamic friction were the more comprehensive of the two, and the paper readily acknowledges that the method presented for computing contact forces with static friction (a Gauss-Seidel-like iterative procedure) was not very reliable. The method also required an approximation for three-dimensional systems (but not for planar systems) that resulted in anisotropic friction. Finally, the results presented did not fully exploit earlier discoveries concerning systems with only dynamic friction, and no static friction.

In this paper, we present a method for computing contact forces with both dynamic and static friction that is considerably more robust than previous methods. Our method requires no approximations for three-dimensional systems, and is much simpler and faster than previous methods. We were extremely surprised to find that our implementation of the method, applied to frictionless systems, was a large improvement compared with the use of large-scale optimization software packages, both in terms of speed and, especially,

SIGGRAPH '94, July 24-29, Orlando, Florida
© ACM 1994 ISBN: 0-89791-667-0 ...$5.00

simplicity.[1] Previous simulation systems for frictionless contact that we know of have used either heuristic solution methods based on linear programming[11], quadratic programming algorithms[7], or constrained linear least-squares algorithms[6]. In all cases the numerical software required is sufficiently complicated that either public-domain or commercially available software packages are required. The problems with this are:

- Serious implementations of linear programming codes are much less common than serious implementations for solving linear systems. Serious implementations for quadratic programming are even rarer.
- A fair amount of mathematical and coding sophistication is required to interface the numerical software package with the simulation software. In some cases, the effort required for an efficient interface was prohibitively high[12].
- The packages obtained contained a large number of adjustable parameters such as numerical tolerances, iteration limits, etc. It is not uncommon for certain contact-force computations to fail with one set of parameters, while succeeding with another, or for a problem to be solvable using one software package, but unsolvable using a different package. In our past work in offline motion simulation, reliability has been a vexing, but tolerable issue: if a given simulation fails to run, one can either alter the initial conditions slightly, hoping to avoid the specific configuration which caused the difficulty, or modify the software itself prior to rerunning the simulation. This approach is clearly not practical in an interactive setting.
- Along the same lines, it is difficult to isolate numerical problems during simulation, because of the complexity of the software packages. Unless great effort is put into understanding the internals of the code, the user is faced with a "black box." This is desirable for black-box code that is bullet-proof, but a serious impediment when the code is not.

Given these hurdles, it is not surprising that analytical methods for systems with contact have not caught on yet. Our recent work has taught us that the difficulties encountered are, in a sense, self-created. In computing contact forces via numerical optimization, we translate a very specific problem (contact-force computation) into a much more general problem (numerical optimization). The translation loses some of the specific structure of the original problem, making the solution task more difficult. The approach we take in this paper is to avoid (as much as possible) abstracting our specific problem into a more general problem. The result is an algorithm that solves a narrower range of problems than general purpose optimization software, but is faster, more reliable, and considerably easier to implement.

3. Contact Model

In this section we will define the structure of the simplest problem we deal with: a system of frictionless bodies contacting at n distinct points. For each contact point $\mathbf{p}_i$ between two bodies, let the scalar a_i denote the relative acceleration between the bodies normal to the contact surface at $\mathbf{p}_i$. (We will not consider the question of impact in this paper; thus, we assume that the relative normal *velocity* of bodies at each contact is zero.) We adopt the convention that a positive acceleration a_i indicates that the two bodies are breaking contact at $\mathbf{p}_i$. Correspondingly, $a_i < 0$ indicates that the bodies are accelerating so as to interpenetrate. An acceleration of $a_i = 0$ indicates that the bodies have zero normal acceleration at $\mathbf{p}_i$ and remain in contact (although the relative tangential acceleration may be nonzero). To prevent interpenetration we require $a_i \geq 0$ for each contact point.

For frictionless systems, the force acting between two bodies at a contact point is normal to the contact surface. We denote the magnitude of the normal force between the bodies at $\mathbf{p}_i$ by the scalar f_i. A positive f_i indicates a repulsive force between the bodies at $\mathbf{p}_i$, while a negative f_i indicates an attractive force. Since contact forces must be repulsive, a necessary condition on f_i is $f_i \geq 0$. Also, since frictionless contact forces are conservative, we must add the condition $f_i a_i = 0$ for each contact point. This condition requires that at least one of f_i and a_i be zero for each contact: either $a_i = 0$ and contact remains, or $a_i > 0$, contact is broken, and f_i is zero.

We will denote the n-vector collection of a_i's as $\mathbf{a}$; the ith element of $\mathbf{a}$ is a_i. The vector $\boldsymbol{f}$ is the collection of the f_i's. (In general, boldface type denotes matrices and vectors; the ith element of a vector $\mathbf{b}$ is the scalar b_i, written in regular type. The symbol $\mathbf{0}$ denotes on appropriately sized vector or matrix of zeros.) The vectors $\mathbf{a}$ and $\boldsymbol{f}$ are linearly related; we can write

$$\mathbf{a} = \mathbf{A}\boldsymbol{f} + \mathbf{b} \tag{1}$$

where $\mathbf{A} \in \mathbf{R}^{n \times n}$ is symmetric and positive semidefinite (PSD), and $\mathbf{b} \in \mathbf{R}^n$ is a vector in the column space of $\mathbf{A}$ (that is, $\mathbf{b} = \mathbf{Ax}$ for some vector $\mathbf{x}$). The matrix $\mathbf{A}$ reflects the masses and contact geometries of the bodies, while $\mathbf{b}$ reflects the external and inertial forces in the system. At any instant of time, $\mathbf{A}$ and $\mathbf{b}$ are known quantities while $\boldsymbol{f}$ is the unknown we are interested in solving for.

The problem of determining contact forces is therefore the problem of computing a vector $\boldsymbol{f}$ satisfying the conditions

$$a_i \geq 0, \quad f_i \geq 0 \quad \text{and} \quad f_i a_i = 0 \tag{2}$$

for each contact point. We will call equation (2) the *normal force conditions*. Using equation (1), we can phrase the problem of determining a suitable $\boldsymbol{f}$ in several forms. First, since f_i and a_i are constrained to be nonnegative, the requirement that $f_i a_i = 0$ for each i is equivalent to requiring that

$$\sum_{i=1}^{n} f_i a_i = \boldsymbol{f}^T \mathbf{a} = 0 \tag{3}$$

since no cancellation can occur. Using equation (1), we can say that $\boldsymbol{f}$ must satisfy the conditions

$$\mathbf{A}\boldsymbol{f} + \mathbf{b} \geq \mathbf{0}, \quad \boldsymbol{f} \geq \mathbf{0} \quad \text{and} \quad \boldsymbol{f}^T(\mathbf{A}\boldsymbol{f} + \mathbf{b}) = 0. \tag{4}$$

Equation (4) defines what is known as a linear complementarity problem (LCP). Thus one solution method for computing contact forces is to formulate and solve the LCP of equation (4). We can also compute contact forces by considering the conditions of equation (2) as a quadratic program (QP): we can equivalently say that a vector $\boldsymbol{f}$ satisfying equation (4) is a solution to the quadratic program

$$\min_{\boldsymbol{f}} \boldsymbol{f}^T(\mathbf{A}\boldsymbol{f} + \mathbf{b}) \quad \text{subject to} \quad \left\{ \begin{array}{c} \mathbf{A}\boldsymbol{f} + \mathbf{b} \geq \mathbf{0} \\ \boldsymbol{f} \geq \mathbf{0} \end{array} \right\}. \tag{5}$$

Phrasing the computation of $\boldsymbol{f}$ as a QP is a natural choice. (The problem of solving QP's has received more attention than the problem of solving LCP's. Both problems are *NP*-hard in general but can be practically solved when $\mathbf{A}$ is PSD.) Having transformed the problem of computing contact forces into a QP, we have a variety of techniques available for solving the QP. Unfortunately, by moving to an optimization problem—minimize $\boldsymbol{f}^T(\mathbf{A}\boldsymbol{f}+\mathbf{b})$—we necessarily lose sight of the original condition $f_i a_i = 0$ for each contact point. Because of this, we are solving a more general, and thus harder,

[1] Actually, not being numerical specialists, any working numerical software we were capable of creating would have to be simpler. We automatically assumed however that such software would be slower than the more comprehensive packages written by experts in the field.

problem than we really need to. In developing an algorithm, we prefer to regard the relationship between $\boldsymbol{f}$ and $\mathbf{a}$ in terms of the n separate conditions $f_i a_i = 0$ in equation (2) rather than the single constraint $\boldsymbol{f}^T\mathbf{a} = 0$ in equation (4) or the minimization of $\boldsymbol{f}^T\mathbf{a}$ in equation (5). In the next section, we describe a physically-motivated method for solving equation (2), along with a practical implementation. Following this, we consider friction in section 5.

4. Frictionless Systems

In this section we present a restriction of our algorithm for computing contact forces with friction to the frictionless case. We also sketch a proof of correctness. We extend the algorithm in section 5 to handle static friction, and dynamic friction in section 6. A description of Dantzig's algorithm for solving LCP's, and an excellent treatment of LCP's in general can be found in Cottle *et al.*[5].

4.1 Algorithm Outline

Dantzig's algorithm for solving LCP's is related to pivoting methods used to solve linear and quadratic programs. The major difference is that all linear and most quadratic programming algorithms begin by first finding a solution that satisfies the constraints of the problem (for us, $\mathbf{A}\boldsymbol{f} + \mathbf{b} \geq \mathbf{0}$ and $\boldsymbol{f} \geq \mathbf{0}$) and then trying to minimize the objective function (for us, $\boldsymbol{f}^T\mathbf{A}\boldsymbol{f} + \boldsymbol{f}^T\mathbf{b}$).

In contrast, Dantzig's algorithm, as applied to the problem of computing contact forces, works as follows. Initially, all contact points but the first are ignored, and f_i is set to zero for all i. The algorithm begins by computing a value for f_1 that satisfies the normal force conditions—equation (2)—for $i = 1$, without worrying about those conditions holding for any other i. Next, the algorithm computes a value for f_2 that satisfies the normal force conditions for $i = 2$ while maintaining the conditions for $i = 1$. This may require modification of f_1. The algorithm continues in this fashion: at any point, the conditions at contact points $1 \leq i \leq k-1$ are satisfied for some k and $f_i = 0$ for $i > k$, and the algorithm determines f_k, possibly altering some of the f_i's for $i < k$, so that the conditions now hold for all $i \leq k$. When the conditions hold for all n contact points, the algorithm terminates.

To make this concrete, imagine that we have so far computed values f_1 through f_{n-1} so that the normal force conditions hold everywhere except possibly at the nth contact point. Suppose that with f_n still set to zero we have $a_n \geq 0$. If so, we immediately have a solution $\boldsymbol{f}$ that satisfies the normal conditions at all n contact points.

Suppose however that for $f_n = 0$ we have $a_n < 0$. Our physical intuition tells us that since we currently have $f_n = 0$, the problem is that the nth contact force is not doing its fair share. We must increase f_n until we reach the point that a_n is zero, and we must do so without violating the normal force conditions at any of the first $n-1$ contact points. Since increasing f_n may change some of the a_i's, we will generally need to modify the other f_i variables as we increase f_n. Our goal is to seek a strength for f_n that is just sufficient to cause a_n to be zero. (We emphasize that this is not a process which takes place over some time interval t_0 to t_1 during a simulation; rather, we are considering the proper value that $\boldsymbol{f}$ should assume at a specific instant in time.)

The adjustments we need to make to f_1 through f_{n-1} as we increase f_n are simple to calculate. Since the order in which contacts are numbered is arbitrary, let us imagine that for the current values of the f_i's we have $a_1 = a_2 = \ldots = a_k = 0$ for some value $0 \leq k \leq n-1$, and for all $k+1 \leq i \leq n-1$, we have $a_i > 0$. Remember that $a_n < 0$. To simplify bookkeeping, we will employ two disjoint index sets C and NC. At this point in the algorithm, let $C = \{1, 2, \ldots, k\}$; thus, $a_i = 0$ for all $i \in C$. Similarly, let $NC = \{k+1, k+2, \ldots, n-1\}$; since $a_i > 0$ for all $i \in NC$, and we have assumed that $f_i a_i = 0$ for $i \leq n-1$, it must be that $f_i = 0$ for all $i \in NC$. Throughout the algorithm, we will attempt to maintain $a_i = 0$ whenever $i \in C$. Similarly, we will try to maintain $f_i = 0$ whenever $i \in NC$. When $i \in C$, we say that the ith contact point is "clamped," and when $i \in NC$ we say the ith contact point is "unclamped." (If i is in neither, the ith contact point is currently being ignored.)

For a unit increase of f_n (that is, if we increase f_n to $f_n + 1$) we must adjust each f_i by some amount Δf_i. Let $\Delta f_n = 1$, and let us set $\Delta f_i = 0$ for all $i \in NC$, since we wish to maintain $f_i = 0$ for $i \in NC$. We wish to choose the remaining Δf_i's for $i \in C$ such that $\Delta a_i = 0$ for $i \in C$. The collection $\Delta\mathbf{a}$ of the Δa_i's is defined by

$$\Delta\mathbf{a} = \mathbf{A}(\boldsymbol{f} + \Delta\boldsymbol{f}) + \mathbf{b} - (\mathbf{A}\boldsymbol{f} + \mathbf{b}) = \mathbf{A}\Delta\boldsymbol{f} \tag{6}$$

where $\Delta\boldsymbol{f}$ denotes the collection of the Δf_i's.

Intuitively, we picture the force f_i at a clamped contact point undergoing some variation in order to maintain $a_i = 0$, while the force at an unclamped contact remains zero. Modifications of this sort will maintain the invariant that $f_i a_i = 0$ for all $1 \leq i \leq n-1$. Since C currently has k elements, computing the unspecified Δf_i's requires solving k linear equations in k unknowns. (In general, C will vary in size during the course of the algorithm. At any point in the algorithm when we are establishing the conditions at the rth contact, C will contain $r-1$ or fewer elements.)

However, we also need to maintain the conditions $f_i \geq 0$ and $a_i \geq 0$. Thus, as we increase f_n, we may find that for some $i \in C$, f_i has decreased to zero. At this point, it may be necessary to unclamp this contact by removing i from C and adding it to NC, so that we do not cause f_i to decrease any further. Conversely, we may find that for some $i \in NC$, a_i has decreased to a value of zero. In this case, we will wish to clamp the contact by moving i from NC into C, preventing a_i from decreasing any further and becoming negative. The process of moving the various indices between C and NC is exactly the numerical process known as pivoting. Given that we start with suitable values for f_1 through f_{n-1}, computing f_n is straightforward. We set $\Delta f_n = 1$ and $\Delta f_i = 0$ for $i \in NC$, and solve for the Δf_i's for $i \in C$ so that $\Delta a_i = 0$ for all $i \in C$. Next, we choose the smallest scalar $s > 0$ such that increasing $\boldsymbol{f}$ by $s\Delta\boldsymbol{f}$ causes either a_n to reach zero, or some index i to move between C and NC. If a_n has reached zero, we are done; otherwise, we change the index sets C and NC, and loop back to continue increasing f_n.

We now describe the process of computing $\Delta\boldsymbol{f}$ along with the step size s. After this, we present the complete algorithm and discuss its properties.

4.2 The Pivot Step

The relation between the vectors $\mathbf{a}$ and $\boldsymbol{f}$ is given by $\mathbf{a} = \mathbf{A}\boldsymbol{f} + \mathbf{b}$. Let us continue with our example in which $C = \{1, 2, \ldots, k\}$ and $NC = \{k+1, k+2, \ldots, n-1\}$. We need to compute $\Delta\boldsymbol{f}$ and then determine how large a multiple of $\Delta\boldsymbol{f}$ we can add to $\boldsymbol{f}$. Currently, we have $a_n < 0$. Let us partition $\mathbf{A}$ and $\Delta\boldsymbol{f}$ by writing

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{v}_1 \\ \mathbf{A}_{12}^T & \mathbf{A}_{22} & \mathbf{v}_2 \\ \mathbf{v}_1^T & \mathbf{v}_2^T & \alpha \end{pmatrix} \quad \text{and} \quad \Delta\boldsymbol{f} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \\ 1 \end{pmatrix} \tag{7}$$

where $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ are square symmetric matrices, $\mathbf{v}_1 \in \mathbf{R}^k$, $\mathbf{v}_2 \in \mathbf{R}^{(n-1)-k}$, α is a scalar, and $\mathbf{x} \in \mathbf{R}^k$ is what we will need to compute. The linear system $\Delta\mathbf{a} = \mathbf{A}\Delta\boldsymbol{f}$ has the form

$$\Delta\mathbf{a} = \mathbf{A}\Delta\boldsymbol{f} = \mathbf{A}\begin{pmatrix} \mathbf{x} \\ \mathbf{0} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{11}\mathbf{x} + \mathbf{v}_1 \\ \mathbf{A}_{12}^T\mathbf{x} + \mathbf{v}_2 \\ \mathbf{v}_1^T\mathbf{x} + \alpha \end{pmatrix}. \tag{8}$$

Since the first k components of $\Delta\mathbf{a}$ need to be zero, we require $\mathbf{A}_{11}\mathbf{x} + \mathbf{v}_1 = \mathbf{0}$; equivalently, we must solve

$$\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1.$$

After solving equation (9), we compute $\Delta\mathbf{a} = \mathbf{A}\Delta\boldsymbol{f}$, and are ready to find the maximum step size parameter s we can scale $\Delta\boldsymbol{f}$ by. For each $i \in C$, if $\Delta f_i < 0$, then the force at the ith contact point is decreasing. The maximum step s we can take without forcing f_i negative is

$$s \leq \frac{f_i}{-\Delta f_i}. \tag{10}$$

Similarly, for each $i \in NC$, if $\Delta a_i < 0$ then the acceleration a_i is decreasing; the maximum step is limited by

$$s \leq \frac{a_i}{-\Delta a_i}. \tag{11}$$

Since we do not wish a_n to exceed zero, if $\Delta a_n > 0$, the maximum step is limited by

$$s \leq \frac{-a_n}{\Delta a_n}. \tag{12}$$

Once we determine s, we increase $\boldsymbol{f}$ by $s\Delta\boldsymbol{f}$, which causes $\mathbf{a}$ to increase by $\mathbf{A}(s\Delta\boldsymbol{f}) = s\Delta\mathbf{a}$. If this causes a change in the index sets C and NC, we make the required change and continue to increase f_n. Otherwise, a_n has achieved zero.

4.3 A Pseudo-code Implementation

The entire algorithm is described below in pseudo-code. The main loop of the algorithm is simply:

```
function compute-forces
    f = 0
    a = b
    C = NC = ∅
    while ∃d  such that  a_d < 0
        drive-to-zero(d)
```

The function *drive-to-zero* increases f_d until a_d is zero. The direction of change for the force, $\Delta\boldsymbol{f}$, is computed by *fdirection*. The function *maxstep* determines the maximum step size s, and the constraint j responsible for limiting s. If j is in C or NC, j is moved from one to the other; otherwise, $j = d$, meaning a_d has been driven to zero, and *drive-to-zero* returns:

```
function drive-to-zero(d)
  L1:
    Δf = fdirection(d)
    Δa = AΔf
    (s, j) = maxstep(f, a, Δf, Δa, d)
    f = f + sΔf
    a = a + sΔa
    if j ∈ C
        C = C − {j}
        NC = NC ∪ {j}
        goto L1
    else if j ∈ NC
        NC = NC − {j}
        C = C ∪ {j}
        goto L1
    else            j must be d, implying a_d = 0
        C = C ∪ {j}
        return
```

The function *fdirection* computes $\Delta\boldsymbol{f}$. We write $\mathbf{A}_{CC}$ to denote the submatrix of $\mathbf{A}$ obtained by deleting the jth row and column of $\mathbf{A}$ for all $j \notin C$. Similarly, $\mathbf{A}_{Cd}$ denotes the dth column of $\mathbf{A}$ with element j deleted for all $j \notin C$. The vector $\mathbf{x}$ represents the change in contact force magnitudes at the clamped contacts. The transfer of $\mathbf{x}$ into $\Delta\boldsymbol{f}$ is the reverse of the process by which elements are removed from the dth column of $\mathbf{A}$ to form $\mathbf{A}_{Cd}$. (That is, for all $i \in C$, we assign to Δf_i the element of $\mathbf{x}$ corresponding to the ith contact.)

```
function fdirection(d)
    Δf = 0              set all Δf_i to zero
    Δf_d = 1
    let A11 = A_CC
    let v1 = A_Cd
    solve A11 x = −v1
    transfer x into Δf
    return Δf
```

Last, the function *maxstep* returns a pair (s, j) with s the maximum step size that can be taken in the direction $\Delta\boldsymbol{f}$ and j the index of the contact point limiting the step size s:

```
function maxstep(f, a, Δf, Δa, d)
    s = ∞
    j = −1
    if Δa_d > 0
        j = d
        s = −a_d/Δa_d
    for i ∈ C
        if Δf_i < 0
            s' = −f_i/Δf_i
            if s' < s
                s = s'
                j = i
    for i ∈ NC
        if Δa_i < 0
            s' = −a_i/Δa_i
            if s' < s
                s = s'
                j = i
    return (s, j)
```

It is clear that if the algorithm terminates, the solution $\boldsymbol{f}$ will yield $a_i \geq 0$ for all i. Since each f_i is initially zero and is prevented from decreasing below zero by *maxstep*, at termination $f_i \geq 0$ for all i. Last, at termination, $f_i a_i = 0$ for all i since either $i \in C$ and $a_i = 0$, or $i \notin C$ and $f_i = 0$.

The only step of the algorithm requiring substantial coding is *fdirection*, which requires forming and solving a square linear system. Remarkably, even if $\mathbf{A}$ is singular (and $\mathbf{A}$ is often extremely rank-deficient in our simulations), the submatrices $\mathbf{A}_{11}$ encountered in the frictionless case are never singular. This is a consequence of $\mathbf{b}$ being in the column space of $\mathbf{A}$.

4.4 Termination of the Algorithm

We will quickly sketch why the algorithm we have described must always terminate, with details supplied in appendix A. Examining the algorithm, the two critical steps are solving $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$ and computing the step size s. First off, could the algorithm fail because it could not compute $\mathbf{x}$? Since $\mathbf{A}$ is symmetric PSD, if $\mathbf{A}$ is nonsingular then $\mathbf{A}_{11}$ is nonsingular and $\mathbf{x}$ exists. Even if $\mathbf{A}$ is singular, the submatrices $\mathbf{A}_{11}$ considered by the algorithm are never singular, as long as $\mathbf{b}$ lies in the column space of $\mathbf{A}$.[2] As a result, the system $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$ can always be solved. This is however a theoretical result. In actual practice, when $\mathbf{A}$ is singular it is possible

[2] A complete proof of this is somewhat involved. The central idea is that if the jth contact point has not yet been considered and represents a "redundant constraint" (that is, adding j into C makes $\mathbf{A}_{11}$ singular) then a_j will not be negative, so there will be no need to call *drive-to-zero* on j. Similarly, if $j \in NC$ and moving j to C would make $\mathbf{A}_{11}$ singular, it will not be the case that a_j tries to decrease below zero, requiring j to be placed in C. Essentially, the nonzero f_i's will do the work of keeping a_j from becoming negative, without f_j having to become positive, allowing j to remain outside of C.

that roundoff errors in the algorithm may cause an index j to enter C so that the resulting matrix $\mathbf{A}_{11}$ is singular. This is a very rare occurrence, but even so, it does not present a practical problem. Appendix A establishes that the vector $\mathbf{v}_1$ is always in the column space of the submatrix $\mathbf{A}_{11}$ arising from *any* index set C. Thus, even if $\mathbf{A}_{11}$ is singular, the equation $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$ is well-conditioned, and is easily solved by standard factorization methods.[3] In essence, we assert that "$\mathbf{A}_{11}$ is never singular, and even if it is, $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$ is still easily solved."

Since it is always possible to solve $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$ and obtain $\Delta\boldsymbol{f}$, the real question of termination must depend on each call to *drive-to-zero* being able to force a_d to zero. To avoid being bogged down in details, let us assume that $\mathbf{A}$ is nonsingular, with specific proofs deferred to appendix A; additionally, appendix A discusses the necessary extensions to cover the case when $\mathbf{A}$ is singular. Although the singular versus the nonsingular cases require slightly different proofs, we emphasize that *the algorithm itself* remains unchanged; that is, the algorithm we have just described works for both positive definite and positive semidefinite $\mathbf{A}$.

The most important question to consider is whether increasing $\boldsymbol{f}$ by an amount $s\Delta\boldsymbol{f}$ actually increases a_d. Given a change $s\Delta\boldsymbol{f}$ in $\boldsymbol{f}$, from equation (8) the increase in a_d is

$$s(\mathbf{v}_1^T\mathbf{x} + \alpha) = s\Delta a_d. \tag{13}$$

Theorem 2 shows that if $\mathbf{A}$ is positive definite, $\mathbf{v}_1^T\mathbf{x} + \alpha$ is always positive. Thus, a_d will increase as long as s is always positive. Since $\mathbf{v}_1^T + \alpha = \Delta a_d > 0$, this shows that *maxstep* never returns with $s = \infty$ and $j = -1$.

Can the algorithm take steps of size zero? In order for *maxstep* to return $s = 0$, it would have to the case that either $f_i = 0$ and $\Delta f_i < 0$ for some $i \in C$ or $a_i = 0$ and $\Delta a_i < 0$ for some $i \in NC$. Theorems 4 and 5 shows that this cannot happen. Thus, s is always positive. Therefore, the only way for a_d to not reach zero is if *drive-to-zero* takes an infinite number of steps $s\Delta\boldsymbol{f}$ that result in in a_d converging to some limit less than or equal to zero. This possibility is also ruled out, since theorem 3 in appendix A shows that the set C of clamped contact points is never repeated during a given call to *drive-to-zero*. Thus, *drive-to-zero* can iterate only a finite number of times before a_d reaches zero.

4.5 Implementation Details

The algorithm just described is very simple to implement and requires relatively little code. The most complicated part involves forming and solving the linear system $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$. This involves some straightforward bookkeeping of the indices in C and NC to correctly form $\mathbf{A}_{11}$ and then distribute the components of $\mathbf{x}$ into $\Delta\boldsymbol{f}$. It is important to note that each call to *fdirection* will involve an index set C that differs from the previous index set C by only a single element. This means that each linear system $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$ will differ from the previous system only by a single row and column. Although each such system can be solved independently (for example, using Cholesky decomposition), for large problems it is more efficient to use an incremental approach.

In keeping with our assertion that nonspecialists can easily implement the algorithm we describe, we note that our initial implementation simply used Gaussian elimination, which we found to be completely satisfactory. (Anticipating the developments of the next section when $\mathbf{A}_{11}$ is nonsymmetrical, we did not bother to use a Cholesky factorization, although this would have performed significantly faster.)

Gill *et al.*[9] describe a package called LUSOL that incrementally factors a sparse matrix $\mathbf{A}$ into the form $\mathbf{A} = \mathbf{LU}$ where $\mathbf{L}$ is lower triangular and $\mathbf{U}$ is upper triangle. Given such a factorization, if $\mathbf{A}$ has dimension n and a new row and column are added to $\mathbf{A}$, or a row and column are eliminated from $\mathbf{A}$, a factorization of the new matrix can be recomputed quickly. Unfortunately, the coding effort for LUSOL is large. One of the authors of the LUSOL package was kind enough to provide us with a modified version of the software[13] that treats $\mathbf{A}$ as a dense matrix and computes a factorization $\mathbf{LA} = \mathbf{U}$ (where $\mathbf{L}$ is no longer triangular). In the dense case, an updated factorization is obtained in $O(n^2)$ time when $\mathbf{A}$ is altered. The modified version contains a reasonably small amount of code. For a serious implementation we highly recommend the use of an incremental factorization routine.

In addition, it is trivial to make the algorithm handle standard bilateral constraints. For a bilateral constraint, we introduce a pair f_i and a_i, and we constrain a_i to always be zero while letting f_i be either positive or negative. Given k such constraints, we initially solve a square linear system of size k to compute compute initial values for all the bilateral f_i's so that all the corresponding a_i's are zero. Each such i is placed into C at the beginning of the algorithm. In *maxstep*, we ignore each index i that is a bilateral constraint, since we do not care if that f_i becomes negative. As a result, the bilateral i's always stay in C and the bilateral a_i's are always zero. Exactly the same modification can be made in the algorithm presented in the next section.

5. Static Friction

The algorithm of the previous section can be considered a constructive proof that there exists a solution $\boldsymbol{f}$ satisfying the normal force conditions for any frictionless system. The algorithm presented in this section grew out of an attempt to prove the conjecture that all systems with static friction, but no dynamic friction, also possess solutions. (The conjecture is false for systems with dynamic friction.) The conjecture currently remains unproven. We cannot prove that the algorithm we present for computing static friction forces will always terminate; if we could, that in itself would constitute a proof of the conjecture. On the other hand, we have not yet seen the algorithm fail, so that the algorithm is at least practical (for the range of simulations we have attempted so far).

Let us consider the situation when there is friction at a contact point. The friction force at a point acts tangential to the contact surface. We will denote the magnitude of the friction force at the ith contact by f_{F_i}, and the magnitude of the relative acceleration in the tangent plane as a_{F_i}. We will also denote the magnitude of the normal force as f_{N_i}, rather than f_i, and the magnitude of the normal acceleration as a_{N_i} rather than a_i. To specify the tangential acceleration and friction force completely in a three-dimensional system, we would also need to specify the *direction* of the acceleration and friction force in the tangent plane. For simplicity, we will begin by dealing with two-dimensional systems. At each contact point, let $\mathbf{t}_i$ be a unit vector tangent to the contact surface; $\mathbf{t}_i$ is unique except for a choice of sign. In a two dimensional system, we will treat f_{F_i} and a_{F_i} as signed quantities. A friction force magnitude of f_{F_i} denotes a friction force of $f_{F_i}\mathbf{t}_i$, and an acceleration magnitude a_{F_i} denotes an acceleration of $a_{F_i}\mathbf{t}_i$. Thus, if a_{F_i} and f_{F_i} have the same sign, then the friction force and tangential acceleration point in the same direction.

Static friction occurs when the relative tangential velocity at a contact point is zero; otherwise, the friction is called dynamic friction. In this section, we will consider only static friction. Any configuration of objects that is initially at rest will have static friction, but no dynamic friction. Additionally, a "first-order" (or quasistatic) simulation world where force and *velocity* are related by $f = mv$ also has static friction but never any dynamic friction

[3]Since $\mathbf{A}_{11}$ is both symmetric and PSD, $\mathbf{A}_{11}$ will still have a Cholesky factorization $\mathbf{A}_{11} = \mathbf{LL}^T$, although $\mathbf{L}$ is singular. Since $\mathbf{L}$ can be simply and reliably computed, this is one possible way of solving for $\mathbf{x}$.

5.1 Static Friction Conditions

At a contact point with static friction, the magnitude v_{Fi} of the relative tangential velocity is zero. If the effect of all the forces in the system produces $a_{Fi} = 0$, meaning that the condition $v_{Fi} = 0$ is being maintained, then f_{F_i} need satisfy only

$$-\mu f_{N_i} \leq f_{F_i} \leq \mu f_{N_i} \tag{14}$$

where the scalar μ denotes the coefficient of friction at the contact point. (We will not bother to index μ over the contact points, although this is easily done.) If the tangential acceleration is not zero, then the conditions on f_{F_i} are more demanding: $|f_{F_i}|$ must be equal to μf_{N_i} and f_{F_i} must have sign opposite that of a_{Fi}.

Following the pattern of section 4, we write that f_{N_i}, a_{Ni}, f_{F_i} and a_{Fi} must satisfy the normal force conditions

$$f_{N_i} \geq 0, \;\; a_{Ni} \geq 0 \;\; \text{and} \;\; f_{N_i} a_{Ni} = 0, \tag{15}$$

as well as

$$|f_{F_i}| \leq \mu f_{N_i}, \; a_{Fi} f_{F_i} \leq 0 \; \text{and} \; a_{Fi}(\mu f_{N_i} - |f_{F_i}|) = 0. \tag{16}$$

The condition $a_{Fi}(\mu f_{N_i} - |f_{F_i}|) = 0$ forces f_{F_i} to have magnitude μf_{N_i} if a_{Fi} is nonzero. The condition $a_{Fi} f_{F_i} \leq 0$ forces a_{Fi} and f_{F_i} to have opposite sign, which means that the friction force opposes the tangential acceleration. We will call the conditions of equation (16) the *static friction conditions*; unless specifically noted, a contact point said to satisfy the static friction conditions implies satisfaction of the normal force conditions as well.

The approach taken by previous attempts[10,3] at modeling static friction has been to form an optimization problem. If we define the quantity scalar z by

$$z = \sum_i \left(|a_{Fi}|(\mu f_{N_i} - |f_{F_i}|) + f_{N_i} a_{Ni} \right) \tag{17}$$

then the problem becomes

$$\min_{f_{N_i}, f_{F_i}} z \text{ subject to } \left\{ \begin{array}{c} f_{N_i} \geq 0 \\ a_{Ni} \geq 0 \end{array} \right\} \text{ and } \left\{ \begin{array}{c} a_{Fi} f_{F_i} \leq 0 \\ |f_{F_i}| \leq \mu f_{N_i} \end{array} \right\}.$$

Computing contact forces in this manner does not appear to be practical.

5.2 Algorithm Outline

We believe it is better to deal with the problem as we did in the frictionless case: as a number of separate conditions. Let us consider the static friction condition with that perspective. We can state the conditions on a_{Fi} and f_{F_i} by considering that the "goal" of the friction force is to keep the tangential acceleration as small as possible, under the restriction $|f_{F_i}| \leq \mu f_{N_i}$. Accordingly, whenever a_{Fi} is nonzero we insist that the friction force do its utmost to "make" a_{Fi} be zero by requiring that the friction force push as hard as possible opposite the tangential acceleration. The reason that we find this a useful characterization is that it is *essentially the same characterization* we employed in section 4 to motivate the development of Dantzig's algorithm.

In section 4.1, we assumed that the normal force conditions were initially met for contacts 1 through $n-1$ and began with $f_{N_n} = 0$. If this resulted in a_{Nn} being nonnegative, then we immediately had a solution. Otherwise, it was in a sense f_{N_n}'s "fault" that a_{Nn} was negative, and we increased f_{N_n} to remedy the situation. We can do *exactly* the same thing to compute static friction forces! Suppose that the first $n-1$ contacts of our system satisfy all the conditions for static friction and that the normal force condition holds for the nth contact point. We set $f_{F_n} = 0$ and consider a_{Nn}. If $n \in NC$, or $n \in C$ but $f_{N_n} = 0$, then the static force condition is trivially met since $|f_{F_n}| = 0 = \mu f_{N_n}$. If not, but it happens that $a_{Fn} = 0$, again, we have satisfied the static friction conditions, since $|f_{F_n}| = 0 < \mu f_{N_n}$. Otherwise, a_{Fn} is nonzero and following our characterization of static friction we must increase the magnitude of the friction force to oppose the tangential acceleration as much as possible.

The procedure to do this is essentially the same as in the frictionless case. Without loss of generality, assume that at the nth contact point $a_{Fn} < 0$. We will gradually increase f_{F_n} while maintaining the static friction and normal conditions at all the other $n-1$ contact points and the normal condition at the nth contact point. As we increase f_{F_n}, at some point, one of two things must happen: either we will reach a point where $f_{F_n} = \mu f_{N_n}$, or we will reach a point where $a_{Fn} = 0$. In either case, the static friction conditions will then be met.

5.3 Maintaining the Static Friction Conditions

Once we have established the static friction conditions at a contact point, we need to maintain them. As before, we maintain the conditions $f_{N_i} \geq 0$, $a_{Ni} \geq 0$ and $f_{N_i} a_{Ni} = 0$ using the index sets C and NC. To maintain the conditions on the f_{F_i} and a_{Fi} variables, we introduce the sets C_F, NC^- and NC^+. The set C_F is analogous to C; whenever $i \in C_F$, we manipulate f_{F_i} to maintain $a_{Fi} = 0$. (We can have $i \in C_F$ and $i \in C$. The fact that $i \in C_F$ means we are maintaining $a_{Fi} = 0$, while the fact that $i \in C$ means we are maintaining $a_{Ni} = 0$.) In contrast to C_F, if $i \in NC^+$, then we have $a_{Fi} < 0$ and $f_{F_i} = \mu f_{N_i}$. As long as $i \in NC^+$, we vary f_{F_i} so that it is always equal to μf_{N_i}. If a_{Fi} becomes zero, we move i from NC^+ into C_F. Thus, NC^+ denotes the set of contacts that have f_{F_i} positive and at the upper bound of μf_{N_i}. Conversely, if $i \in NC^-$, then we have $a_{Fi} > 0$ and $f_{F_i} = -\mu f_{N_i}$. Again, as long as $i \in NC^-$ we will maintain the condition $f_{F_i} = -\mu f_{N_i}$, and move i into C_F if a_{Ni} becomes zero. Whenever we are increasing some f_{N_d} or increasing or decreasing some f_{F_d}, computing the corresponding changes in the other f_{F_i} and f_{N_i} variables, along with the maximum possible step size, is exactly the same as in the previous section.

In the frictionless case, when we managed to drive a_{Nd} to zero, we added d into C. For static friction, if the driving process stops because a_{Fd} has reached zero, we insert d into C_F. Otherwise, the process stopped because $|f_{F_d}| = \mu f_{N_d}$ and we add d into NC^- or NC^+ as appropriate. Before we present our algorithm for computing static friction forces in two dimensions, we discuss why the algorithm we present is not guaranteed to terminate.

5.4 Algorithm Correctness

In section 4, we showed that as we increased f_d, the acceleration a_d always increased in response, guaranteeing that a sufficiently large increase of f_d would achieve $a_d = 0$. We also showed that the index set C would never repeat while forcing a particular a_d to zero, guaranteeing we would not converge to some negative value. Finally, we showed that steps of size zero would not occur, guaranteeing that we would always make progress towards $a_d = 0$. For static friction, we can show all these properties except for the last.

First, let us show that if we start with $a_{Fd} < 0$, as we increase f_{F_d}, either we will reach a point where $f_{F_d} = \mu f_{N_d}$, or we will reach a point where $a_{Fd} = 0$. This is not obvious. Since f_{N_d} is nonzero (otherwise $f_{F_d} = 0$ would satisfy the static friction conditions), we must have $d \in C$. This means that as we increase f_{F_d}, we may also be requiring that f_{N_d} change as well. If μf_{N_d} increases faster than f_{F_d} does, then f_{F_d} will never reach a value of μf_{N_d}.

Similarly, it is not necessarily the case that increasing f_{F_d} will cause a_{Fd} to increase. The reason for this is the following: the relation between the acceleration variables and force variables is still linear, and we can write

$$\mathbf{a} = \begin{pmatrix} a_{N1} \\ a_{F1} \\ \vdots \\ a_{Nn} \\ a_{Fn} \end{pmatrix} = \mathbf{A} \begin{pmatrix} f_{N1} \\ f_{F1} \\ \vdots \\ f_{Nn} \\ f_{Fn} \end{pmatrix} + \mathbf{b} = \mathbf{A}\boldsymbol{f} + \mathbf{b} \qquad (18)$$

where $\mathbf{A} \in \mathbf{R}^{2n \times 2n}$ and $\mathbf{b} \in \mathbf{R}^{2n}$ and $\boldsymbol{f}$ and $\mathbf{a}$ are the collection of the f and a variables. As long as we have no dynamic friction, it is still the case that $\mathbf{A}$ is symmetric and PSD. For a unit increase in f_{F_d}, we solve for Δf_{N_i} and Δf_{F_i} exactly as we did in section 4. That is, for $i \in C$, we require $\Delta a_{N_i} = 0$, and for all other i, we have $\Delta f_{N_i} = 0$. For the friction forces, almost the same holds: for $i \in C_F$ we require $\Delta a_{F_i} = 0$. However, for $i \in NC^-$, instead of setting $\Delta f_{F_i} = 0$, we require $\Delta f_{F_i} = -\mu \Delta f_{N_i}$, to maintain $f_{F_i} = -\mu f_{N_i}$. Similarly, for $i \in NC^+$ we require $\Delta f_{F_i} = \mu \Delta f_{N_i}$ to maintain $f_{F_i} = \mu f_{N_i}$. The side conditions $\Delta f_{F_i} = \pm\mu \Delta f_{N_i}$ prevent us from applying theorem 2 as we did in section 4 and claiming that a_{F_d} increases as f_{F_d} increases. In fact, in some situations, increasing f_{F_d} will cause a_{F_d} to *decrease*. The same holds for f_{N_d} as well; prior to working on f_{F_d} we may find that increasing f_{N_d} to establish the normal force conditions may cause causes a_{N_d} to decrease.

Is it possible then that we can drive some f_{F_d} or f_{N_d} infinitely far without reaching a stopping point? Fortunately, it is not. Theorem 3 of appendix A states that for frictionless systems, as we increase f_{N_i} the index set C never repeats. Exactly the same theorem is trivially extended to cover static friction. Thus, we will never encounter exactly the same sets C, NC, C_F, NC^- and NC^+ while driving a given f_{N_n} or f_{F_n} variable. We can use this to show that increasing f_{N_d} will eventually cause a_{N_d} to increase. Exactly the same argument shows that increasing f_{F_d} eventually causes a_{F_d} to increase.

THEOREM 1 *In a problem with static friction only, if $a_{N_d} < 0$ and $f_{N_d} = 0$ hold initially, a large enough increase in f_{N_d} will eventually force a_{N_d} to increase.*

PROOF. Suppose that we could arbitrarily increase f_{N_d} without causing a_{N_d} to increase. Since $\mathbf{A}$ is positive definite, in light of theorem 2 this can only happen if one or more of the side conditions $\Delta f_{F_i} = \pm\mu \Delta f_{N_i}$ hold, implying that $NC^- \cup NC^+ \neq \emptyset$. Since the index sets C, NC, C_F, NC^- and NC^+ never repeat, there are only finitely many combinations of those sets that can be encountered while increasing f_{N_d}. That means that we can only undergo finitely many changes of the sets while increasing f_{N_d}. Eventually, we settle into a state where we can increase f_{N_d} without a_{N_d} increasing and without any change occurring in the index sets.

However, this cannot be, because of the definition of the index sets. For $i \in C$, to avoid a change in index sets, we must have $\Delta f_{N_i} \geq 0$; otherwise, a sufficiently large step will move i into NC. The same logic requires that for $i \in NC$ we must have $\Delta a_{N_i} \geq 0$, otherwise a_{N_i} will fall to zero. This yields $\Delta f_{N_i} \Delta a_{N_i} = 0$ for all i. For the friction forces, if $i \in C_F$, then $\Delta a_{F_i} = 0$ so $\Delta a_{F_i} \Delta f_{F_i} = 0$. For $i \in NC^+$, we have $a_{F_i} < 0$, requiring $\Delta a_{F_i} \leq 0$ to avoid having to move i from NC^+ to C_F. Since $\Delta f_{N_i} \geq 0$ for all i and $\Delta f_{F_i} = \mu \Delta f_{N_i}$, we have $\Delta f_{F_i} \geq 0$. This yields $\Delta a_{F_i} \Delta f_{F_i} \leq 0$ for all $i \in NC^+$. A symmetric argument holds, yielding $\Delta a_{F_i} \Delta f_{F_i} \leq 0$ for all $i \in NC^-$.

Additionally, for at least one i in NC^- or NC^+, both Δa_{F_i} and Δf_{F_i} are nonzero; otherwise, we could remove each side condition $\Delta f_{F_i} = \pm\mu \Delta f_{N_i}$ and add the conditions $\Delta f_{F_i} = 0$ and $\Delta a_{F_i} = 0$ without altering any other Δf_{N_i} or Δf_{F_i}. If we did so however, we would then be entitled to apply theorem 2, contradicting our assumption that a_{N_d} is nonincreasing. Thus, for at least one i we have $\Delta a_{F_i} \Delta f_{F_i}$ strictly less than zero. Combining that with the fact that $\Delta a_{N_i} \Delta f_{N_i} \leq 0$ and $\Delta a_{F_i} \Delta f_{F_i} \leq 0$ for all i we obtain

$$\sum_i^n \Delta a_{N_i} \Delta f_{N_i} + \sum_i^n \Delta a_{F_i} \Delta f_{F_i} = \Delta \mathbf{a}^T \Delta \boldsymbol{f} < 0. \qquad (19)$$

Since $\Delta \mathbf{a} = \mathbf{A} \Delta \boldsymbol{f}$, this gives us

$$\Delta \mathbf{a}^T \Delta \boldsymbol{f} = \Delta \boldsymbol{f}^T \mathbf{A} \Delta \boldsymbol{f} < 0. \qquad (20)$$

Since $\Delta \boldsymbol{f}$ is nonzero and $\mathbf{A}$ is PSD, this is a contradiction (even if $\mathbf{A}$ is singular). Thus, f_{N_d} cannot be increased without bound without eventually causing a_{N_d} to increase. □

However, there is still the possibility of taking steps of size zero, and this is something that can and does occur when running the algorithm. Theorems 4 and 5 may fail to hold because of the side conditions $\Delta f_{F_i} = \pm\mu \Delta f_{N_i}$. The following scenario is possible: for some $i \in C$, f_{N_i} decreases to zero. Accordingly, i is moved from C to NC. Upon computing $\Delta \boldsymbol{f}$ with the new index set, we may find that $\Delta a_{N_i} < 0$ (which is ruled out in the frictionless case by theorem 4). As a result, a step of size zero is taken, and i is moved *back* into C. Clearly, the algorithm settles into a loop, alternately moving i between C and NC by taking a step of size zero each time. We cannot rule this behavior out in our algorithm for static friction. (This is also our current sticking point in trying to prove the conjecture that all systems with only static friction have solutions.) Fortunately, we have found a practical remedy for the problem.

While attempting to establish the normal force or static friction conditions at some point k, if we observe that a variable i is alternating between C and NC (or between NC^- and C_F or NC^+ and C_F), we remove i from *both* C and NC (or from C_F and NC^- or NC^+). Temporarily, we will "give up" trying to maintain the normal or static friction conditions at the ith contact point. We do so at the expense of making "negative progress," in the sense that although we will have achieved our immediate goal (establishing normal or friction conditions at a particular contact point), we will have done so by sacrificing normal and/or static friction conditions previously achieved at other contacts. The algorithm will be forced to reestablish the conditions at the points we have given up on at some later time. Since contact points no long necessarily keep their static friction or normal force conditions once established, we cannot prove (as yet) that this process will ever terminate.

We have however used this algorithm on a large variety of problems, and we have never yet encountered any situation for which our algorithm went into an infinite loop. We speculate that either no such situation is possible, meaning that all systems with static friction have solutions, or it requires an extremely carefully constructed problem to cause our algorithm to loop (although the latter possibility does not necessarily imply that there is in fact no solution $\boldsymbol{f}$). A third possibility of course is that we simply have not sufficiently exercised our simulation system.

5.5 Algorithm for Computing Static Friction Forces

We now describe the necessary modifications to Dantzig's algorithm to handle static friction forces. The modifications increase the complexity of the "logical" portion of the algorithm, but the heart of the numerical code, computing $\Delta \boldsymbol{f}$, remains the same. We give a description of the necessary modifications of each procedure of the algorithm.

Modifications to *compute-frictionless-forces*

The sets C, NC, C_F, NC^+, and NC^- are all initially empty. The main loop continually scans for a contact point at which the normal or static friction conditions are not met. If no such points exist, the algorithm terminates, otherwise, *drive-to-zero* is called to establish the conditions. Note that one must first establish the normal force conditions at a given point before establishing the static friction conditions there. In the event that the algorithm gives up on a contact point i which has the normal conditions established, it will do so because f_{N_i} is oscillating between C and NC. At this point $f_{N_i} = 0$, and the normal conditions can be reestablished later.

If however we give up on the static friction conditions at the ith contact point, f_{F_i} may be nonzero. (We cannot discontinuously set f_{F_i} to zero as this might break the conditions at *all* the other contact points.) Later, when the algorithm attempts to reestablish the static friction conditions at i, we first drive f_{F_i} to zero (simply by instructing *drive-to-zero* to increase or decrease f_{F_i} until $f_{F_i} = 0$).

Modifications to *drive-to-zero*

This function is the same, except that there are more ways for the index sets to change. If the limiting constraint j returned by *maxstep* is the index of the force being driven, j is moved into C if it represents a normal force, and otherwise into C_F, NC^-, or NC^+ as appropriate; the procedure then returns. Otherwise, j is moved between C and NC if it represents a normal force, and otherwise between C_F and NC^- or NC^+ as appropriate. If j attempts to move into a set it just came from, and the previous step size was zero, j is removed from whatever index set it was in. This is the point at which the algorithm temporarily gives up on maintaining the conditions at the jth contact point.

Modifications to *fdirection*

The modifications are minor. First, if we are driving a normal force, we set $\Delta f_{N_d} = 1$, otherwise we set $\Delta f_{F_d} = \pm 1$, depending which way we want to drive the force. The index sets establish the set of equations to solve: for $i \in NC$, we set $\Delta f_{N_i} = 0$; for $i \in C$ we require $\Delta a_{N_i} = 0$; for $i \in C_F$ we require $\Delta a_{F_i} = 0$; and for $i \in NC^+ \cup NC^-$ we require $\Delta f_{F_i} = \pm \Delta f_{N_i}$.

Modifications to *maxstep*

The modifications here are obvious. For each member j in an index set, we compute the minimum step size s that causes j to need to change to another set. For the driving index d, we compute the step size that causes us to reach $a_{N_d} = 0$ for a normal force, and $a_{F_d} = 0$ or $f_{F_d} = \pm \mu f_{N_d}$ for a friction force. The minimum step s that can be taken, along with the constraint j responsible for that limit, is returned.

5.6 Three-dimensional Systems

We have been assuming that our system is two-dimensional. The extension to three dimensions is straightforward.At each contact point, let us denote vectors $\mathbf{u} \in \mathbf{R}^3$ tangent to the contact surface as pairs (x, y) by choosing a local coordinate system such that $(1, 0)$ and $(0, 1)$ denote an orthonormal pair of tangent vectors. Let (a_{x_i}, a_{y_i}) and (f_{x_i}, f_{y_i}) denote the relative tangential acceleration and friction force, respectively, at the ith contact point. In three dimensions, the Coulomb friction law requires that the friction force be at least partially opposed to the tangential acceleration; that is,

$$(f_{x_i}, f_{y_i}) \cdot (a_{x_i}, a_{y_i}) = f_{x_i} a_{x_i} + f_{y_i} a_{y_i} \leq 0. \tag{21}$$

The optimization approach taken in previous work[10,3] makes enforcing $|f_{F_i}| \leq \mu f_{N_i}$ difficult, because

$$|f_{F_i}| = (f_{x_i}^2 + f_{y_i}^2)^{\frac{1}{2}} \leq \mu f_{N_i} \tag{22}$$

is a nonlinear constraint. However, this constraint is easily dealt with by our algorithm. In place of the two sets NC^- and NC^+, for three-dimensional systems, we use a single set NC_F. In two dimensions, given Δf_{N_i} and Δf_{F_i}, determining the step size s so that $f_{F_i} + s\Delta f_{F_i} = \mu(f_{N_i} + s\Delta f_{N_i})$ is trivial. In three dimensions, computing $s > 0$ so that

$$(f_{x_i} + s\Delta f_{x_i})^2 + (f_{y_i} + s\Delta f_{y_i})^2 = (\mu(f_{N_i} + s\Delta f_{N_i}))^2 \tag{23}$$

is also trivial. As a result, it is easy to augment *maxstep* to move i into NC_F when $f_{x_i}^2 + f_{y_y}^2 = (\mu f_{N_i})^2$ and also easy to detect when to move i back into C_F. When i moves into NC_F, we record the direction that the friction force is pointing in. As long as i remains in NC_F, we require the friction force (f_{x_i}, f_{y_i}) to maintain the same direction it had when i most recently entered NC_F. Once i moves back into C_F, the pair (f_{x_i}, f_{y_i}) may point in any direction.

To initially establish the static friction conditions for f_{x_i} and f_{y_i}, we first increase f_{x_i} (assuming $a_{x_i} < 0$) until either i moves into NC_F, or a_{x_i} reaches zero. If i is in NC_F, we are done, otherwise, we now adjust f_{y_i} so that either a_{y_i} reaches zero, or i moves into NC_F. Reversing the order with which one considers x and y, or rotating the local coordinate system in the tangent plane may give rise to different solutions of $\boldsymbol{f}$ with this method. This is a consequence of the condition of equation (21), which does not completely specify the direction of friction when the tangential acceleration is nonzero at a contact point.

6. Dynamic Friction

If the relative tangential velocity at a contact point is nonzero, then dynamic friction occurs, as opposed to static friction. Regardless of the resulting tangential acceleration, the strength of the friction force satisfies

$$|f_{F_i}| = \mu f_{N_i}, \tag{24}$$

with the direction of the force exactly opposite the relative tangential velocity. Since f_{F_i} is no longer an independent variable, when we formulate equation (18), we can replace all occurences of f_{F_i} with $\pm \mu f_{N_i}$. This replacement results in a matrix $\mathbf{A}$ which is unsymmetric and possibly indefinite as well. Because of this, systems with dynamic friction can fail to have solutions for the contact force magnitudes, requiring the application of an impulsive force. Another consequence of $\mathbf{A}$ losing symmetry and definiteness is that all the theorems in this paper which require $\mathbf{A}$ to be symmetric and PSD fail to hold. Remarkably, this turns out to be a fortunate development.

Previously, Baraff[3] presented an algorithm for computing friction forces and impulses for systems with dynamic friction but no static friction; the intent was to treat the problem of nonexistence of a solution $\boldsymbol{f}$. Baraff's method for computing either regular or impulsive forces for systems with dynamic friction involved using Lemke's algorithm[5] for solving LCP's. It is noted that Lemke's algorithm can terminate by encountering an "unbounded ray." The algorithm we have just presented for static friction requires absolutely no modifications to handle dynamic friction in this manner. An unbounded ray corresponds to finding a state in which one can drive a variable f_{N_i} or f_{F_i} to infinity without forcing a_{N_i} or a_{F_i} to zero, or inducing a change in the index sets C, NC, C_F, NC^+ or NC^-. When this occurs, it is easily detected, in that *maxstep* returns a step size of $s = \infty$. Note that theorem 2 tells us that an infinite step cannot occur if $\mathbf{A}$ is symmetric and PSD. which means that infinite steps are possible only if there is dynamic friction in the system. Either our algorithm finds a solution $\boldsymbol{f}$, or at some point $s = \infty$, and the current force direction $\Delta \boldsymbol{f}$ matches the definition proposed by Baraff for suitably applying impulsive forces to systems with dynamic friction. As a result, we can unify our treatment of both dynamic and static friction in a single algorithm. We note in closing that we feel that this is mostly a theoretical, and not a practical concern, because we have encountered this infinite driving mostly in situations where μ has been made unrealistically large.

7. Results

Our method for computing contact and friction forces is both reliable and fast. Like most pivoting algorithms (for example, the simplex algorithm for linear programming), worst-case problems resulting in exponential running times can be constructed. Empirically however, the algorithm appears to require about $O(n)$ calls to *drive-to-zero* for systems with and without friction. Our real interest however is the performance of the algorithm in actual practice.

We have implemented the two-dimensional algorithm for static friction in an interactive setting and the three-dimensional algorithm in an offline simulation system. For frictionless systems, our solution algorithm compares favorably to Gaussian elimination with partial pivoting. Given a matrix $\mathbf{A}$ and vector $\mathbf{b}$, the algorithm of section 4 takes *only* two to three times longer to compute the contact forces than it would take to solve the linear system $\mathbf{Ax} = \mathbf{b}$, using Gaussian elimination. Compared with the best QP methods we know of, our algorithm runs five to ten times faster, on problems up to size $n = 150$. For systems with friction, there is no comparable solution algorithm we can compare our algorithm to.

Interactive simulations of $2\frac{1}{2}$D mechanisms are shown in figures 1 and 2. Fixed objects are colored in black. Objects in different "levels" are different colors (orange, purple, and green) and have no collision interaction. White circles indicate a bilateral point-to-point constraint. In figure 2, the green circles indicate contact points. Both systems can be simulated robustly at a consistent framerate of 20–30Hz on an SGI R4400 workstation.

Acknowledgements

This research was funded in part by an NSF Research Initiation Award and an AT&T Foundation Equipment Grant. We would like to sincerely thank Michael Saunders and Richard Cottle for supplying us with a dense version of the LUSOL package and for clarifying several technical and historical points about LCP's.

Appendix A: Theorems

In this appendix, we prove some theorems necessary to show that the algorithm for frictionless contact forces in section 4 terminates. For simplicity, we consider only the case when $\mathbf{A}$ is nonsingular and sketch the modifications necessary if $\mathbf{A}$ is singular.

THEOREM 2 *Let the symmetric positive definite matrix* $\mathbf{A}$ *be partitioned as in equation (7). If* $\mathbf{x}$ *satisfies* $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$*, then the quantity* $\mathbf{v}_1^T\mathbf{x} + \alpha$ *is always positive.*

PROOF. Principal submatrices of $\mathbf{A}$ are positive definite, so $\alpha > 0$, $\mathbf{A}_{11}$ is positive definite and the submatrix

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{v}_1 \\ \mathbf{v}_1^T & \alpha \end{pmatrix}$$

is positive definite. Applying a Cholesky factorization, we can write

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{v}_1 \\ \mathbf{v}_1^T & \alpha \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11} & \mathbf{0} \\ \mathbf{L}_{12}^T & \mathbf{L}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{11}^T & \mathbf{L}_{12} \\ \mathbf{0} & \mathbf{L}_{22} \end{pmatrix} \tag{25}$$

where $\mathbf{L}_{11}$ and $\mathbf{L}_{12}$ have the same dimensions as $\mathbf{A}_{11}$ and $\mathbf{v}_1$ respectively, and $\mathbf{L}_{22}$ is a positive scalar. Note that since $\mathbf{A}_{11} = \mathbf{L}_{11}\mathbf{L}_{11}^T$ is invertible, $\mathbf{L}_{11}$ is also invertible and $\mathbf{A}_{11}^{-1} = \mathbf{L}_{11}^{-T}\mathbf{L}_{11}^{-1}$. From equation (25), we have $\mathbf{v}_1 = \mathbf{L}_{11}\mathbf{L}_{12}$. Since $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$, we also have $\mathbf{x} = -\mathbf{A}_{11}^{-1}\mathbf{v}_1$. Then

$$\begin{aligned} \mathbf{v}_1^T\mathbf{x} + \alpha &= \alpha - \mathbf{v}_1^T\mathbf{A}_{11}^{-1}\mathbf{v}_1 \\ &= \alpha - (\mathbf{L}_{12}^T\mathbf{L}_{11}^T)\mathbf{A}_{11}^{-1}(\mathbf{L}_{11}\mathbf{L}_{12}) \\ &= \alpha - \mathbf{L}_{12}^T\mathbf{L}_{11}^T\mathbf{L}_{11}^{-T}\mathbf{L}_{11}^{-1}\mathbf{L}_{11}\mathbf{L}_{12} \\ &= \alpha - \mathbf{L}_{12}^T\mathbf{L}_{12}. \end{aligned}$$

From equation (25) we have $\alpha = \mathbf{L}_{12}^T\mathbf{L}_{12} + \mathbf{L}_{22}^2$; thus

$$\mathbf{v}_1^T\mathbf{x} + \alpha = \alpha - \mathbf{L}_{12}^T\mathbf{L}_{12} = \mathbf{L}_{22}^2. \tag{26}$$

Since $\mathbf{L}_{22}$ is positive, $\mathbf{v}_1^T\mathbf{x} + \alpha$ is positive. □

Almost the same result applies when $\mathbf{A}$ is not invertible. In this case, $\mathbf{A}_{11}$ may be singular; note however that a Cholesky factorization can still be obtained although $\mathbf{L}_{11}$ may now be singular. Since it is still the case that $\mathbf{A}_{11} = \mathbf{L}_{11}\mathbf{L}_{11}^T$, and $\mathbf{L}_{11}$ and $\mathbf{L}_{11}\mathbf{L}_{11}^T$ have exactly the same column space, the fact that $\mathbf{v}_1 = \mathbf{L}_{11}\mathbf{L}_{12}$ implies that $\mathbf{v}_1$ is in the column space of $\mathbf{A}_{11}$. Thus, the equation $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$ will always have a solution. Using basic continuity principles[4] it can be shown that in the singular case, $\mathbf{v}_1^T\mathbf{x} + \alpha \geq 0$.

THEOREM 3 *During a given call to drive-to-zero, the same index set C is never repeated.*

PROOF. Suppose some index set C was repeated during a call to *drive-to-zero*. Since $C \cup NC$ remains constant during a given invocation of *drive-to-zero* (except at the last step, where the driving index d is added to C), whenever C is repeated, NC is repeated as well. Let the values of $\boldsymbol{f}$ the first time and second time C is encountered be denoted $\boldsymbol{f}^{(1)}$ and $\boldsymbol{f}^{(2)}$ respectively. Let $\mathbf{a}^{(1)} = \mathbf{A}\boldsymbol{f}^{(1)} + \mathbf{b}$ and $\mathbf{a}^{(2)} = \mathbf{A}\boldsymbol{f}^{(2)} + \mathbf{b}$. The intuition of the proof is simple: if the algorithm could have increased $\boldsymbol{f}$ along a straight line from $\boldsymbol{f}^{(1)}$ to $\boldsymbol{f}^{(2)}$, it would have done so. The fact that it did not means that increasing from $\boldsymbol{f}^{(1)}$ to $\boldsymbol{f}^{(2)}$ must have required a change between C and NC. We show that this cannot happen because of the inherent convexity involved, contradicting the fact that C was repeated.

Specifically, we have $a_i^{(1)} = a_i^{(2)} = 0$ for all $i \in C$ and $a_i^{(1)} \geq 0$ and $a_i^{(2)} \geq 0$ for all $i \in NC$. Given C and NC, the vector $\boldsymbol{f}$ is increased in the direction $\Delta\boldsymbol{f}$ where $\Delta f_i = 0$ for $i \in NC$, $\Delta f_d = 1$ and $\Delta a_i = 0$ for $i \in C$. However, the vector

$$\mathbf{y} = \frac{\boldsymbol{f}^{(2)} - \boldsymbol{f}^{(1)}}{f_d^{(2)} - f_d^{(1)}} \tag{27}$$

fulfills all the conditions for $\Delta\boldsymbol{f}$, since $y_d = 1$, $y_i = 0$ for $i \in NC$, and the vector

$$\mathbf{Ay} = \frac{\mathbf{A}(\boldsymbol{f}^{(2)} - \boldsymbol{f}^{(1)})}{f_d^{(2)} - f_d^{(1)}} = \frac{\mathbf{a}^{(2)} - \mathbf{a}^{(1)}}{f_d^{(2)} - f_d^{(1)}} \tag{28}$$

has its ith component equal to zero for all $i \in C$. Thus, when C was first encountered, $\Delta\boldsymbol{f} = \mathbf{y}$ was chosen. If $a_d = 0$ could have been achieved by increasing $\boldsymbol{f}$ in this direction, *drive-to-zero* would have terminated, and C would not have been repeated. This means that in increasing from $\boldsymbol{f}^{(1)}$ in the direction $\Delta\boldsymbol{f} = \mathbf{y}$, it was necessary to change C and NC prior to reaching $\boldsymbol{f}^{(2)}$; that is for some value t in the range $0 < t < 1$, either

$$\left(\mathbf{A}(\boldsymbol{f}^{(1)} + t(\boldsymbol{f}^{(2)} - \boldsymbol{f}^{(1)})) + \mathbf{b}\right)_j < 0 \tag{29}$$

for some $j \in NC$ or

$$\left(\boldsymbol{f}^{(1)} + t(\boldsymbol{f}^{(2)} - \boldsymbol{f}^{(1)})\right)_j < 0 \tag{30}$$

for some $j \in C$. However, since neither of the above two equations are satisfied when $t = 0$ or $t = 1$, and the equations involve only

[4]If $\mathbf{A}$ is a symmetric PSD singular matrix, then there exist arbitrarily small perturbation matrices ϵ such that $\mathbf{A} + \epsilon$ is symmetric positive definite (and hence nonsingular).

linear relations and inequalities, by convexity, neither of the two above equations are satisfied for any value $0 < t < 1$. This contradicts the assumption that the same set C was encountered twice during a call of *drive-to-zero*. □

This theorem also extends to the algorithm for static friction in section 5. Namely, we claim that the index sets C, NC, C_F, NC^- and NC^+ are never repeated while driving a given force variable f_{N_d} or f_{F_d}. The proof is exactly the same, the only difference being that extra conditions of the form $\Delta f_{F_i} = \pm\mu\Delta f_{N_i}$ may be present. However, given that $\boldsymbol{f}^{(1)}$ and $\boldsymbol{f}^{(2)}$ satisfy these extra conditions, any vector $\boldsymbol{f}^{(1)} + t(\boldsymbol{f}^{(2)} - \boldsymbol{f}^{(1)})$ for $0 < t < 1$ will satisfy these properties as well. Again, this means that the algorithm should have gone directly from $\boldsymbol{f}^{(1)}$ to $\boldsymbol{f}^{(2)}$, contradicting the fact that the index sets were repeated.

The last two theorems guarantee that the frictionless algorithm never takes steps of size zero, as long as the system is not degenerate. A *degenerate problem* (not to be confused with **A** being singular) is one that would require the algorithm to to make two or more changes in the index sets C and NC at exactly the same time (for example, if two normal forces decreased to zero simultaneously). When degeneracy occurs, it is possible that some number of size zero steps are taken. Cottle[5, section 4.2, pages 248–251] proves that the frictionless algorithm cannot loop due to degeneracy.

Proving that a nondegenerate problem never takes steps of size zero is relatively straightforward. We need to show that whenever $i \in C$ moves to NC, a_i immediately increases. As a result, i cannot immediately move back to C without taking a step of nonzero size. Similarly, we need to show that whenever $i \in NC$ moves to C, f_i immediately increases.

THEOREM 4 *In a nondegenerate problem, when an index i moves from C to NC, a_i immediately increases.*

PROOF. Without loss of generality, let $C = \{1, 2, ..., k-1\}$ and let us assume that the kth contact has just moved from C to NC. When k was still in C, we computed Δf_i by solving the system $\mathbf{A}_{11}\mathbf{x} = -\mathbf{v}_1$ and setting $\Delta f_i = x_i$. Let $\mathbf{A}_{11}$ and $\mathbf{x}$ be partitioned by

$$\mathbf{A}_{11}\mathbf{x} = \begin{pmatrix} \mathbf{B} & \mathbf{w} \\ \mathbf{w}^T & \beta \end{pmatrix}\begin{pmatrix} \mathbf{u} \\ y \end{pmatrix} = \begin{pmatrix} \mathbf{z} \\ c \end{pmatrix} = -\mathbf{v}_1 \qquad (31)$$

where $\mathbf{B} \in \mathbf{R}^{(k-1)\times(k-1)}$, $\mathbf{u}, \mathbf{w}, \mathbf{z} \in \mathbf{R}^k$ and y, β, and c are scalars. This yields

$$\mathbf{u} = \mathbf{B}^{-1}(\mathbf{z} - \mathbf{w}y) \qquad \text{and} \qquad \mathbf{w}^T\mathbf{u} = c - \beta y \qquad (32)$$

or

$$\mathbf{w}^T\mathbf{B}^{-1}(\mathbf{z} - \mathbf{w}y) = c - \beta y. \qquad (33)$$

Since this $\Delta\boldsymbol{f}$ caused f_k to decrease to zero, $\Delta f_k = y$ must have been negative.

Once k moves into NC and we recompute $\Delta\boldsymbol{f}$, we need to show the new Δa_k will be positive. Let $\tilde{\mathbf{u}}$ and $\tilde{y}$ denote the new values computed for $\mathbf{u}$ and y when we resolve for $\Delta\boldsymbol{f}$. Since k is now in NC, we set $\Delta f_k = \tilde{y} = 0$, and solve

$$\mathbf{B}\tilde{\mathbf{u}} + \mathbf{w}\tilde{y} = \mathbf{z} \qquad (34)$$

to obtain

$$\tilde{\mathbf{u}} = \mathbf{B}^{-1}\mathbf{z}. \qquad (35)$$

From equations (8) and (31), the new Δa_k is

$$\Delta a_k = \mathbf{w}^T\tilde{\mathbf{u}} + \beta\tilde{y} - c = \mathbf{w}^T\tilde{\mathbf{u}} - c. \qquad (36)$$

Substituting from equations (35) and (33), we have

$$\begin{aligned} \Delta a_k &= \mathbf{w}^T\mathbf{B}^{-1}\mathbf{z} - c \\ &= -\mathbf{w}^T\mathbf{B}^{-1}\mathbf{w}y - \beta y \\ &= -y(\mathbf{w}^T\mathbf{B}^{-1}\mathbf{w} + \beta). \end{aligned} \qquad (37)$$

Since $\mathbf{A}_{11}$ is positive definite, $\mathbf{B}^{-1}$ is positive definite, and β is positive, so $\mathbf{w}^T\mathbf{B}^{-1}\mathbf{w} + \beta$ must be positive. Since y is negative, $-y$ is positive, and we conclude that $\Delta a_k > 0$. □

This theorem extends immediately to the case when **A** is singular, because the index sets C encountered never produce a singular submatrix $\mathbf{A}_{11}$.

THEOREM 5 *In a nondegenerate problem, when an index i moves from NC to C, f_i immediately increases.*

PROOF. The proof is constructed in the same way as the proof of the previous theorem. □

References

[1] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *Computer Graphics (Proc. SIGGRAPH)*, volume 23, pages 223–232. ACM, July 1989.

[2] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In *Computer Graphics (Proc. SIGGRAPH)*, volume 24, pages 19–28. ACM, August 1990.

[3] D. Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10:292–352, 1993.

[4] R.W. Cottle and G.B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and its Applications*, 1:103–125, 1968.

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

[6] P. Gill, S. Hammarling, W. Murray, M. Saunders, and M. Wright. User's guide for LSSOL: A Fortran package for constrained linear least-squares and convex quadratic programming. Technical Report Sol 86-1, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1986.

[7] P. Gill, W. Murray, M. Saunders, and M. Wright. User's guide for QPSOL: A Fortran package for quadratic programming. Technical Report Sol 84-6, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1984.

[8] P. Gill, W. Murray, M. Saunders, and M. Wright. User's guide for NPSOL: A Fortran package for nonlinear programming. Technical Report Sol 86-2, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1986.

[9] P.E. Gill, W. Murray, M.A. Saunders, and H.W. Wright. Maintaining LU factors of a general sparse matrix. *Linear Algebra and its Applications*, 88/89:239–270, 1987.

[10] P. Lötstedt. Numerical simulation of time-dependent contact friction problems in rigid body mechanics. *SIAM Journal of Scientific Statistical Computing*, 5(2):370–393, 1984.

[11] R.E. Marsten. The design of the XMP linear programming library. *ACM Transactions on Mathematical Software*, 7(4):481–497, 1981.

[12] B. Murtagh and M. Saunders. MINOS 5.1 User's guide. Technical Report Sol 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1987.

[13] M. Saunders. Personal communication. September 1993.

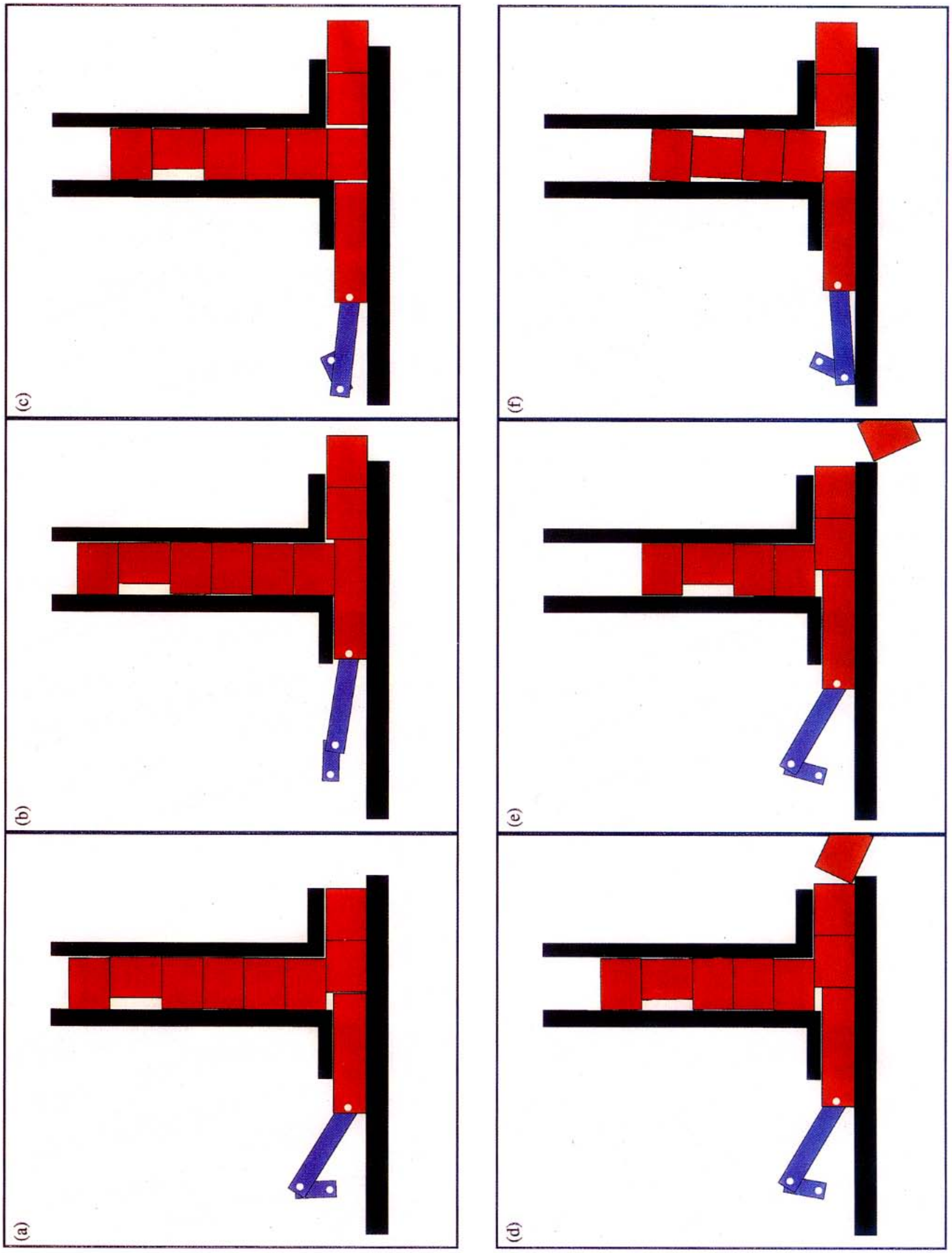

Figure 1: Time-lapse simulation sequence of a blockfeeder.

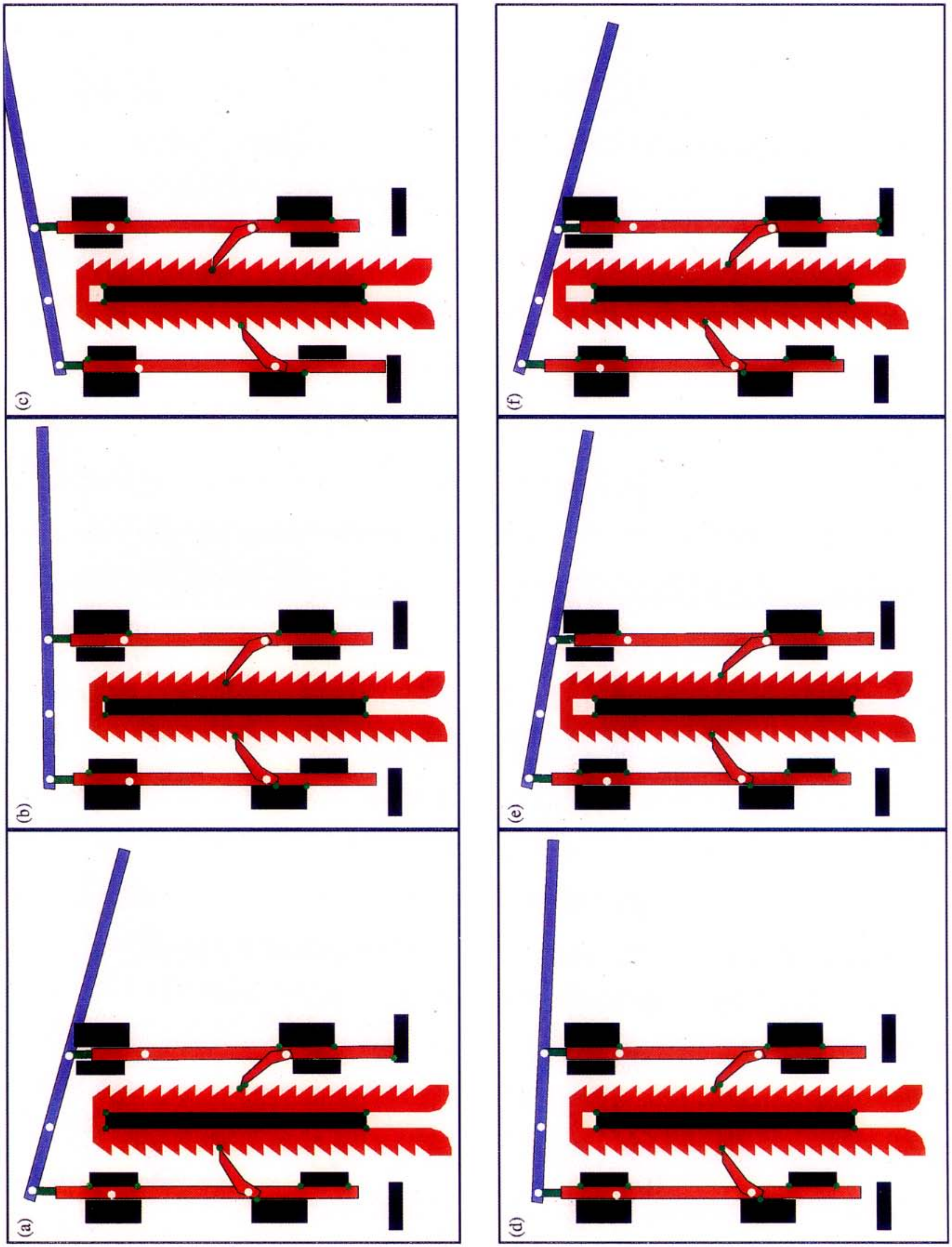

Figure 2: Time-lapse simulation sequence of a double-action jack.