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\cite{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 simultaneous 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\cite{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\cite{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 systems 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\cite{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

Löstedt\cite{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\cite{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,
simplicity.\(^1\) Previous simulation systems for frictionless contact that we know of have used either heuristic solution methods based on linear programming\(\cite{11}\), quadratic programming algorithms\(\cite{7}\), or constrained linear least-squares algorithms\(\cite{6}\). 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\(\cite{12}\).
- 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 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 \(p_i\) between two bodies, let the scalar \(a_i\) denote the relative acceleration between the bodies normal to the contact surface at \(p_i\). (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 \(p_i\). 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 \(p_i\) 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 \(p_i\) by the scalar \(f_i\). A positive \(f_i\) indicates a repulsive force between the bodies at \(p_i\), 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 \(f_i = 0\).

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

\[
a = A f + b\tag{1}
\]

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

The problem of determining contact forces is therefore the problem of computing a vector \(f\) satisfying the conditions

\[
a_i \geq 0, \ \ f_i \geq 0 \ \ \text{and} \ \ f_i a_i = 0 \tag{2}
\]

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 \(f\) 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

\[
\sum_{i=1}^{n} f_i a_i = f^T a = 0 \tag{3}
\]

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

\[
A f + b \geq 0, \ \ f \geq 0 \ \ \text{and} \ \ f^T (A f + b) = 0. \tag{4}
\]

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 \(f\) satisfying equation (4) is a solution to the quadratic program

\[
\min f^T (A f + b) \ \ \text{subject to} \ \ \begin{cases} A f + b & \geq 0 \\ f & \geq 0 \end{cases}. \tag{5}
\]

Phrasing the computation of \(f\) 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 \(A\) 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 \(f^T (A f + b)\)—we necessarily lose sight of the original condition \(f a_i = 0\) for each contact point. Because of this, we are solving a more general, and thus harder,
problem than we really need to. In developing an algorithm, we prefer to regard the relationship between \( f \) and \( a \) in terms of the \( n \) separate conditions \( f_i a_i = 0 \) in equation (2) rather than the single constraint \( f a = 0 \) in equation (4) or the minimization of \( f^T a \) 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, \( A f + b \geq 0 \) and \( f \geq 0 \)) and then trying to minimize the objective function (for us, \( f^T A f + f^T b \)).

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 \( f_{k+1} \), 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 \( f_i \) through \( f_{n-1} \) so that the normal force conditions hold everywhere except possibly at the \( n \)th contact point. Suppose that with \( f_n \) still set to zero we have \( a_0 \geq 0 \). If so, we immediately have a solution \( f \) that satisfies the normal conditions at all \( n \) contact points.

Suppose however that for \( f_n = 0 \) we have \( a_0 < 0 \). Our physical intuition tells us that since we currently have \( f_n = 0 \), the problem is that the \( n \)th contact force is not doing its fair share. We must increase \( f_n \) until we reach the point that \( a_0 \) 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 \( a_0 \) to be zero. (We emphasize 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 \( f \) 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 \( a_1 = a_2 = \ldots = a_0 = 0 \) for some values \( 0 \leq k \leq n - 1 \), and for all \( k + 1 \leq i \leq n - 1 \), we have \( a_i > 0 \). Remember that \( a_0 < 0 \). 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 \( f_{NC} = 0 \) 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 \( i \)th contact point is “clamped,” and when \( i \in NC \) we say the \( i \)th contact point is “unclamped.” (If \( i \) is in neither, the \( i \)th 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 \( f_j \) by some amount \( \Delta f \). Let \( \Delta f = 0 \), and let us set \( \Delta f = 0 \) for all \( i \in NC \), since we wish to maintain \( f_i = 0 \) for \( i \in NC \). We wish to choose the remaining \( \Delta f_i \)’s for \( i \in C \) such that \( \Delta a_i = 0 \) for \( i \in C \). The collection \( \Delta a \) of the \( \Delta a_i \)’s is defined by

\[
\Delta a = (f + \Delta f) + b - (A f + b) = A \Delta f
\]

where \( \Delta f \) denotes the collection of the \( \Delta f_i \)’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 \( f a_i = 0 \) for all \( 1 \leq i \leq n - 1 \). Since \( C \) currently has \( k \) elements, computing the unspecified \( \Delta f_i \)’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 \( r \)th 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 \( r \) 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 \( r \) 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 \( f_i \) through \( f_{r-1} \), computing \( f_r \) is straightforward. We set \( \Delta f_r = 0 \) and \( \Delta f_i = 0 \) for \( i \in NC \), and solve for the \( \Delta f_i \)’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 \( f \) by \( s \Delta f \) causes either \( a_i \) to reach zero, or some index \( i \) to move between \( C \) and \( NC \). If \( a_i \) has reached zero, we are done; otherwise, we change the index sets \( C \) and \( NC \), and loop back to continue increasing \( f_r \).

We now describe the process of computing \( \Delta f \) 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 \( a \) and \( f \) is given by \( a = Af + b \). 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 \( \Delta f \) and then determine how large a multiple of \( \Delta f \) we can add to \( f \). Currently, we have \( a_0 < 0 \). Let us partition \( A \) and \( \Delta f \) by writing

\[
A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}, \quad \Delta f = \begin{bmatrix} x \\ 1 \end{bmatrix}
\]

(7)

where \( A_{11} \) and \( A_{22} \) are square symmetric matrices, \( v_1 \in \mathbb{R}^1 \), \( v_2 \in \mathbb{R}^{(n-1)-k} \), \( \alpha \) is a scalar, and \( x \in \mathbb{R}^1 \) is what we will need to compute. The linear system \( \Delta a = A \Delta f \) has the form

\[
\Delta a = A \Delta f = \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} A_{12}x + v_1 \\ A_{22}x + v_2 \end{bmatrix}.
\]

(8)

Since the first \( k \) components of \( \Delta a \) need to be zero, we require \( A_{11}x + v_1 = 0 \); equivalently, we must solve

\[
A_{11}x = -v_1.
\]

(9)
After solving equation (9), we compute $\Delta a = A \Delta f$, and are ready to find the maximum step size parameter $s$ we can scale $\Delta f$ by. For each $i \in C$, if $\Delta f_i < 0$, then the force at the $i$th 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 $a_i$ to exceed zero, if $\Delta a_i > 0$, the maximum step is limited by

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

Once we determine $s$, we increase $f$ by $s \Delta f$, which causes $a$ to increase by $A(s \Delta f) = s \Delta a$. If this causes a change in the index sets $C$ and $NC$, we make the required change and continue to increase $f_i$. Otherwise, $a_i$ 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:

```plaintext
function compute-forces
    $f = 0$
    $a = b$
    $C = NC = \emptyset$
    while 3$d$ such that $a_d < 0$
        drive-to-zero($d$)
    return
```

The function `drive-to-zero` increases $f$ until $a_d$ is zero. The direction of change for the force, $\Delta f$, is computed by `direction`. 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:

```plaintext
function drive-to-zero($d$)
    $L_1$: $\Delta f = \text{direction}($d$)$
    $\Delta a = A \Delta f$
    $(s, j) = \text{maxstep}($f, $a, \Delta f, \Delta a, d$$)$
    $f = f + s \Delta f$
    $a = a + s \Delta a$
    if $j \in C$
        $C = C - \{j\}$
        $NC = NC \cup \{j\}$
        goto $L_1$
    else if $j \in NC$
        $NC = NC - \{j\}$
        $C = C \cup \{j\}$
        goto $L_1$
    else
        $j$ must be $d$, implying $a_d = 0$
        $C = \emptyset$
    return
```

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

```plaintext
function direction($d$)
    $\Delta f = 0$
    $\Delta f_d = 1$
    let $A_{d1} = A_{Cd}$
    let $v_i = A_{d1}$
    solve $A_{d1} x = -v_i$
    transfer $x$ into $\Delta f$
    return $\Delta f$
```

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

```plaintext
function maxstep($f$, $a$, $\Delta f$, $\Delta a$, $d$)
    $s = \infty$
    $j = -1$
    if $\Delta a_d > 0$
        $j = d$
        $s = -a_d/\Delta a_d$
    for $i \in C$
        if $\Delta f_i < 0$
            $s' = -f_i/\Delta f_i$
        else
            $s' = s$
        if $s' < s$
            $s = s'$
        $j = i$
    for $i \in NC$
        if $\Delta a_i < 0$
            $s' = -a_i/\Delta a_i$
        else
            $s' = s$
        if $s' < s$
            $s = s'$
        $j = i$
    return $(s, j)$
```

It is clear that if the algorithm terminates, the solution $f$ 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 `direction`, which requires forming and solving a square linear system. Remarkably, even if $A$ is singular (and $A$ is often extremely rank-deficient in our simulations), the submatrices $A_{11}$ encountered in the frictionless case are never singular. This is a consequence of $b$ being in the column space of $A$.

### 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 $A_{11} x = -v_i$ and computing the step size $s$. First off, could the algorithm fail because it could not compute $x$? Since $A$ is symmetric PSD, if $A$ is nonsingular then $A_{11}$ is nonsingular and $x$ exists. Even if $A$ is singular, the submatrices $A_{11}$ considered by the algorithm are never singular, as long as $b$ lies in the column space of $A$.\footnote{A complete proof of this is somewhat involved. The central idea is that if the $j$th contact point has not yet been considered and represents a "redundant constraint" (that is, adding $j$ into $C$ makes $A_{11}$ singular) then $a_j$ will not be negative, so there will be no need to call `drive-to-zero` on $j$. Similarly, if $j \notin C$ and moving $j$ to $C$ would make $A_{11}$ 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 $a_i$ from becoming negative, without $f_j$ having to become positive, allowing $j$ to remain outside of $C$.} As a result, the system $A_{11} x = -v_i$ can always be solved. This is however a theoretical result. In actual practice, when $A$ is singular it is possible
that roundoff errors in the algorithm may cause an index $i$ to enter $C$ so that the resulting matrix $A_{11}$ is singular. This is a very rare occurrence, but even so, it does not present a practical problem. Appendix A establishes that the vector $v_i$ is always in the column space of the submatrix $A_{11}$ arising from any index set $C$. Thus, even if $A_{11}$ is singular, the equation $A_{11}v = -v_i$ is well-conditioned, and is easily solved by standard factorization methods. In essence, we assert that “$A_{11}$ is never singular, and even if it is, $A_{11}v = -v_i$ is still easily solved.”

Since it is always possible to solve $A_{11}x = -v_i$ and obtain $\Delta f_i$, the real question of termination must depend on each call to drive-to-zero being able to force $a_i$ to zero. To avoid being bogged down in details, let us assume that $A$ is nonsingular, with specific proofs deferred to appendix A; additionally, appendix A discusses the necessary extensions to cover the case when $A$ 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 $A$.

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

$$s(v_i^T x + a_i) = s\Delta a_i.$$  \hfill (13)

Theorem 2 shows that if $A$ is positive definite, $v_i^T x + a_i$ is always positive. Thus, $a_i$ will increase as long as $s$ is always positive. Since $v_i^T x + a_i = \Delta a_i > 0$, this shows that maxstep never returns with $s = \infty$ and $f = -1$.

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 $a_i$ to not reach zero is if drive-to-zero takes an infinite number of steps $\Delta a_i$ that result in $a_i$ 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 $a_i$ 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 $A_{11}x = -v_i$. This involves some straightforward bookkeeping of the indices in $C$ and $NC$ to correctly form $A_{11}$ and then distribute the components of $x$ into $\Delta f_i$. It is important to note that each call to $\text{fdirection}$ 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 $A_{11}x = -v_i$ 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 $A_{11}$ is nonsymmetrical, we did not bother to use a Cholesky factorization, although this would have performed significantly faster.)

Gill et al.\cite{9} describe a package called LU SOL that incrementally factors a sparse matrix $A$ into the form $A = LU$, where $L$ is lower triangular and $U$ is upper triangle. Given such a factorization, if $A$ has dimension $n$ and a new row and column are added to $A$, or a row and column are eliminated from $A$, 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\cite{13} that treats $A$ as a dense matrix and computes a factorization $LA = U$ (where $L$ is no longer triangular). In the dense case, an updated factorization is obtained in $O(n^2)$ time when $A$ 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.

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 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 $f$ 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 $i$th contact by $f_i$, and the magnitude of the relative acceleration in the tangent plane as $a_i$. We will also denote the magnitude of the normal force as $f_N$, rather than $f_i$, and the magnitude of the normal acceleration as $a_N$, 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 $t$ be a unit vector tangent to the contact surface; $t$ is unique except for a choice of sign. In a two dimensional system, we will treat $f_i$ and $a_i$ as signed quantities. A friction force magnitude of $f_i$ denotes a friction force of $f_i t$, and an acceleration magnitude $a_i$ denotes an acceleration of $a_i t$. Thus, if $a_i$ and $f_i$ 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.
5.1 Static Friction Conditions
At a contact point with static friction, the magnitude \( v_F \) of the relative tangential velocity is zero. If the effect of all the forces in the system produces \( a_F = 0 \), meaning that the condition \( v_F = 0 \) is being maintained, then \( F_F \) need satisfy only

\[
-\mu F_N \leq F_F \leq \mu F_N
\]

where the scalar \( \mu \) denotes the coefficient of friction at the contact point. (We will not bother to index \( \mu \) over the contact points, although this is easily done.) If the tangential acceleration is not zero, then the conditions on \( F_F \) are more demanding: \( |F_F| \) must be equal to \( \mu F_N \), and \( F_F \) must have opposite sign of that of \( a_F \).

Following the pattern of section 4, we write that \( F_N, a_N, F_F \) and \( a_F \) must satisfy the normal force conditions

\[
f_N \geq 0, \quad a_N \geq 0 \quad \text{and} \quad f_N a_N = 0
\]

as well as

\[
|F_F| \leq \mu F_N, \quad a_F F_F \leq 0 \quad \text{and} \quad a_F (\mu F_N - |F_F|) = 0.
\]

The condition \( a_F (\mu F_N - |F_F|) = 0 \) forces \( F_F \) to have magnitude \( \mu F_N \) if \( a_F \) is nonzero. The condition \( a_F F_F \leq 0 \) forces \( F_F \) 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

\[
z = \sum_i \left( |F_F| (\mu F_N - |F_F|) + F_N a_N \right)
\]

then the problem becomes

\[
\min_{f_N, a_N, F_F} z \text{ subject to } \begin{cases} f_N \geq 0, \\ a_N \geq 0 \end{cases} \quad \text{and} \quad \begin{cases} a_F F_F \leq 0 \\ |F_F| \leq \mu F_N \end{cases}.
\]

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 \( a_F \) and \( F_F \) by considering that the “goal” of the friction force is to keep the tangential acceleration as small as possible, under the restriction \( |F_F| \leq \mu F_N \). Accordingly, whenever \( a_F \) is nonzero we insist that the friction force do its utmost to “make” \( a_F \) 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 \( f_{Nn} = 0 \). If this resulted in \( a_{Nn} \) being negative, then we immediately had a solution. Otherwise, it was in a sense \( f_{Nn} \)’s “fault” that \( a_{Nn} \) was negative, and we increased \( f_{Nn} \) 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 \( f_{Nn} = 0 \) and consider \( a_{Nn} \). If \( n \in \text{NC} \), or \( n \in \text{C} \) but \( f_{Nn} = 0 \), then the static force condition is trivially met since

\[
|f_{Nn}| = 0 = \mu f_{Nn}.
\]

If not, but it happens that \( a_{Fn} = 0 \), again, we have satisfied the static friction conditions, since \( |f_{Nn}| < \mu f_{Nn} \).

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 \( f_{Fn} \) 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 \( f_{Fn} \), at some point, one of two things must happen: either we will reach a point where \( f_{Fn} = \mu f_{Nn} \), 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

\[
\begin{align*}
|f_{Nn}| &\leq \mu f_{Nn}, \\
a_{Nn} &\leq 0 \\
a_{Fn} (\mu f_{Nn} - |f_{Fn}|) &\leq 0,
\end{align*}
\]

The condition \( a_{Fn} (\mu f_{Nn} - |f_{Fn}|) = 0 \) forces \( F_F \) to have magnitude \( \mu f_{Nn} \) if \( a_{Fn} \) is nonzero. The condition \( a_{Fn} F_F \leq 0 \) forces \( F_F \) 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

\[
z = \sum_i \left( |F_F| (\mu f_{Nn} - |F_F|) + f_{Nn} a_{Nn} \right)
\]

then the problem becomes

\[
\min_{f_{Nn}, a_{Nn}, F_F} z \text{ subject to } \begin{cases} f_{Nn} \geq 0, \\ a_{Nn} \geq 0 \end{cases} \quad \text{and} \quad \begin{cases} a_{Fn} F_F \leq 0 \\ |F_F| \leq \mu f_{Nn} \end{cases}.
\]

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

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

Similarly, it is not necessarily the case that increasing \( f_{Fd} \) 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

\[ \mathbf{a} = \begin{pmatrix} a_{N_1} \\ a_{F_1} \\ \vdots \\ a_{N_n} \\ a_{F_n} \end{pmatrix} = A \begin{pmatrix} f_{N_1} \\ f_{F_1} \\ \vdots \\ f_{N_n} \\ f_{F_n} \end{pmatrix} + \mathbf{b} = A \mathbf{f} + \mathbf{b} \quad (18) \]

where \( A \in \mathbb{R}^{2n \times 2n} \) and \( \mathbf{b} \in \mathbb{R}^n \) and \( \mathbf{f} \) 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 \( A \) is symmetric and PSD. For a unit increase in \( f_{F} \), we solve for \( \Delta f_{N} \) and \( \Delta f_{F} \) 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 \( i \) for \( a \) for \( i \) /50.

For the friction forces, if \( a \) for \( i \) for \( 1 \) /22.

The side conditions \( \Delta f_{N_i} = \pm \mu \Delta f_{N_i} \) prevent us from applying theorem 2 as we did in section 4 and claiming that \( a_{F_i} \) increases. In fact, in some situations, increasing \( f_{F_i} \) will cause \( a_{F_i} \) to decrease. The same holds for \( f_{N} \) as well; prior to working on \( f_{F} \), we may find that increasing \( f_{F} \) to establish the normal force conditions may cause causes \( a_{F} \) to decrease.

Is it possible then that we can drive some \( f_{N} \) or \( f_{F} \) infinitely far without reaching a stopping point? Fortunately, it is not. Theorem 3 of appendix A states that for frictionless systems, as we increase \( f_{N} \), 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 \( C, NC, CF, NC^{-}, \) and \( NC^{+} \) while driving a given \( f_{N} \) or \( f_{F} \) variable. We can use this to show that increasing \( f_{N} \) will eventually cause \( a_{F} \) to increase. Exactly the same argument shows that increasing \( f_{F} \) eventually causes \( a_{N} \) to increase.

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

**PROOF.** Suppose that we could arbitrarily increase \( f_{N} \) without causing \( a_{F} \) to increase. Since \( A \) is positive definite, in light of theorem 2 this can only happen if one or more of the side conditions \( \Delta f_{N_i} = \pm \mu \Delta f_{N_i} \) hold, implying that \( NC^{-} \cup NC^{+} \neq \emptyset \). Since the inductive hypothesis \( C, NC, CF, NC^{-}, \) and \( NC^{+} \) never repeat, there are only finitely many combinations of those sets that can be encountered while increasing \( f_{N} \). That means that we can only undergo finitely many changes of the sets while increasing \( f_{N} \). Eventually, we settle into a state where we can increase \( f_{N} \) without altering 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 \( \Delta a_{F} \geq 0 \); otherwise \( a_{F} \) will to zero. This yields \( \Delta f_{N_i} + \Delta a_{F} = 0 \) for all \( i \).

For the friction forces, if \( i \in C \), then \( \Delta a_{F} = 0 \) so \( \Delta a_{F} + \Delta f_{F} = 0 \). For \( i \in NC \), we have \( a_{F} < 0 \), requiring \( \Delta a_{F} \leq 0 \) to avoid having to move \( i \) from \( NC^{-} \) to \( CF \). Since \( \Delta f_{N_i} \geq 0 \) for all \( i \) and \( \Delta f_{F} = \mu \Delta f_{N_i} \), we have \( \Delta f_{F} \geq 0 \). This yields \( \Delta a_{F} + \Delta f_{F} \leq 0 \) for all \( i \in NC^{+} \). A symmetric argument holds, yielding \( \Delta a_{F} + \Delta f_{F} \leq 0 \) for all \( i \in NC^{-} \).

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

\[ \sum_{i} \Delta a_{F} + \Delta f_{F} = \Delta a \mathbf{f} < 0. \quad (19) \]

Since \( \Delta a = A \Delta f \), this gives us

\[ \Delta a \mathbf{f} < 0. \quad (20) \]

Since \( \Delta f \) is nonzero and \( A \) is PSD, this is a contradiction (even if \( A \) is singular). Thus, \( f_{N} \) cannot be increased without bound without eventually causing \( a_{F} \) 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 \( \Delta f_{N_i} = \pm \mu \Delta f_{N_i} \). The following scenario is possible: for some \( i \in C \), \( f_{N} \) decreases to zero. Accordingly, \( i \) is moved from \( C \) to \( NC \). Upon computing \( \Delta f \) with the new index set, we may find that \( \Delta a_{F} \geq 0 \) (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 \( CF \) or \( NC^{+} \)), we remove \( i \) from both \( C \) and \( NC \) (or from \( CF \) and \( NC^{-} \) or \( NC^{+} \)). Temporarily, we will “give up” trying to maintain the normal or static friction conditions at the \( k \)th 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 \( f \)). 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 \( \Delta f \), remains the same. We give a description of the necessary modifications of each procedure of the algorithm.

**Modifications to compute-frictionless-forces**

The sets \( C, NC, CF, NC^{-}, \) 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, \( \text{drive-to-zero} \) 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 \( i \)th 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 \( C \) between \( C \) and \( NC \) if it represents a normal force, and otherwise into \( NC \), \( NC^{-} \), 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 \( C \), \( NC^{-} \) 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 \( i \)th contact point.

Modifications to \( f_{direction} \)

The modifications are minor. First, if we are driving a normal force, we set \( \Delta f_{N,i} = 1 \), otherwise we set \( \Delta f_{F,i} = \pm 1 \), 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 \( \Delta f_{N,i} = 0 \); for \( i \in C \) we require \( \Delta f_{F,i} = 0 \); 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 \( \Delta s_{Nd} = 0 \) for a normal force, and \( \Delta s_{Nd} = 0 \) or \( \Delta s_{Fd} = \pm \mu f_{N,i} \) 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 \( u \in \mathbb{R}^{3} \) 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 \( (a_{i}, a_{h}) \) and \( (f_{i}, f_{h}) \) denote the relative tangential acceleration and friction force, respectively, at the \( j \)th 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,

\[
(f_{i}, f_{h}) \cdot (a_{i}, a_{h}) = f_{i}a_{i} + f_{h}a_{h} \leq 0.
\]

The optimization approach taken in previous work[10,3] makes enforcing \( |f_{F,i}| \leq \mu f_{N,i} \) difficult, because

\[
|f_{F,i}| = (f_{i}^{2} + f_{h}^{2})^{2} \leq \mu f_{N,i}
\]

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 \( NC \). In two dimensions, given \( \Delta f_{N} \), and \( \Delta f_{F} \), determining the step size \( s \) so that \( f_{N,i} + s \Delta f_{N,i} = \mu (f_{N,i} + s \Delta f_{N,i}) \) is trivial. In three dimensions, computing \( s > 0 \) so that

\[
(f_{i} + s \Delta f_{i})^{2} + (f_{h} + s \Delta f_{h})^{2} = (\mu (f_{N,i} + s \Delta f_{N,i}))^{2}
\]

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

To initially establish the static friction conditions for \( f_{N} \) and \( f_{F} \), we first increase \( f_{N} \) (assuming \( a_{N,i} < 0 \)) until either \( i \) moves into \( NC \), or \( a_{N,i} \) reaches zero. If \( i \) is in \( NC \), we are done, otherwise, we now adjust \( f_{N,i} \) so that either \( a_{N,i} \) reaches zero, or \( i \) moves into \( NC \). 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 \( f \) 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

\[
|f_{F,i}| = \mu f_{N,i},
\]

with the direction of the force exactly opposite the relative tangential velocity. Since \( f_{F} \) is no longer an independent variable, when we formulate equation (18), we can replace all occurrences of \( f_{F} \) with \( \pm \mu f_{N,i} \). This replacement results in a matrix \( A \) which is symmetric 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 \( A \) losing symmetry and definiteness is that all the theorems in this paper which require \( A \) 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 \( f \). 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 \( f_{F} \) to infinity without forcing \( a_{F,i} \) or \( a_{N,i} \) to zero, or inducing a change in the index sets \( C, NC, CR, NC^{-} \)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 \( A \) 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 \( f \), or at some point \( s = \infty \), and the current force direction \( \Delta f \) 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 \( \mu \) 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 algorithms compares favorably to Gaussian elimination with partial pivoting. Given a matrix $A$ and vector $b$, 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 $Ax = b$, 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-D 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 20–30 Hz 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 LCPs.

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 $A$ is nonsingular and sketch the modifications necessary if $A$ is singular.

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

**PROOF.** Principal submatrices of $A$ are positive definite, so $\alpha > 0$, $A_{ii}$ is positive definite and the submatrix

$$
\begin{pmatrix}
A_{ii} & v_i \\
v_i^T & \alpha
\end{pmatrix}
$$

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

$$
\begin{pmatrix}
A_{ii} & v_i \\
v_i^T & \alpha
\end{pmatrix} =
\begin{pmatrix}
L_{ii} & 0 \\
L_{ii}^T & L_{ii2}
\end{pmatrix}
\begin{pmatrix}
L_{ii}^T & L_{i12} \\
0 & L_{22}
\end{pmatrix}
$$

where $L_{ii}$ and $L_{ii2}$ have the same dimensions as $A_{ii}$ and $v_i$ respectively, and $L_{22}$ is a positive scalar. Note that since $A_{ii} = L_{ii}L_{ii}^T$ is invertible, $L_{ii}$ is also invertible and $A_{ii}^{-1} = L_{ii}^{-1}L_{ii}^{-T}$. From equation (25), we have $v_i = L_{ii}^{-1}L_{i12}$. Since $A_{ii}x = -v_i$, we also have $x = -A_{ii}^{-1}v_i$. Then

$$
v_i^T x + \alpha = -v_i^T A_{ii}^{-1}v_i = -\alpha - (L_{i12}^T L_{i12})^{-1}L_{i12}^T L_{i12}
$$

From equation (25) we have $\alpha = L_{i12}^T L_{i12} + L_{22}^T$; thus

$$
v_i^T x + \alpha = \alpha - L_{i12}^T L_{i12} = L_{22}^T.
$$

Since $L_{22}$ is positive, $v_i^T x + \alpha$ is positive. □

Almost the same result applies when $A$ is not invertible. In this case, $A_{ii}$ may be singular; note however that a Cholesky factorization can still be obtained although $L_{ii}$ may now be singular. Since it is still the case that $A_{ii} = L_{ii}L_{ii}^T$, and $L_{ii}$ and $L_{ii2}$ have exactly the same column space, the fact that $v_i = L_{ii}L_{i12}$ implies that $v_i$ is in the column space of $A_{ii}$. Thus, the equation $A_{ii}x = -v_i$ will always have a solution. Using basic continuity principles it can be shown that in the singular case, $v_i^T x + \alpha \geq 0$.

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 $f$ the first time and second time $C$ is encountered be denoted $f^{(1)}$ and $f^{(2)}$ respectively. Let $a^{(1)} = Af^{(1)} + b$ and $a^{(2)} = Af^{(2)} + b$. The intuition of the proof is simple: if the algorithm had increased $f$ along a straight line from $f^{(1)}$ to $f^{(2)}$, it would have done so. The fact that it did not means that increasing from $f^{(1)}$ to $f^{(2)}$ 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 $a^{(1)} = a^{(2)} = 0$ for all $i \in C$ and $a^{(1)} \geq 0$ and $a^{(2)} \geq 0$ for all $i \in NC$. Given $C$ and $NC$, the vector $f$ is increased in the direction $\Delta f$ where $\Delta f_i = 0$ for $i \in NC$, $\Delta f_i = 1$ and $\Delta a_i = 0$ for $i \in C$. However, the vector

$$
y = \frac{f^{(2)} - f^{(1)}}{f^{(2)}_d - f^{(1)}_d}
$$

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

$$
Ay = \frac{Af^{(2)} - Af^{(1)}}{f^{(2)}_d - f^{(1)}_d} = \frac{a^{(2)} - a^{(1)}}{f^{(2)}_d - f^{(1)}_d}
$$

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

$$
(A(f^{(1)} + tf^{(2)} - f^{(1)}) + b) < 0
$$

for some $j \in NC$ or

$$
(f^{(1)} + tf^{(2)} - f^{(1)}) < 0
$$

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

4If $A$ is a symmetric PSD singular matrix, then there exist arbitrarily small perturbation matrices $\epsilon$ such that $A + \epsilon$ 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, Cr, NC^r and NC^v are never repeated while driving a given force variable f_{x_k} or f_{y_k}. The proof is exactly the same, the only difference being that extra conditions of the form Δ f_{x_k} = ±μΔ f_{y_k} may be present. However, given that f^{(1)} and f^{(2)} satisfy these extra conditions, any vector f^{(1)} + f^{(2)} for 0 < t < 1 will satisfy these properties as well. Again, this means that the algorithm should have gone directly from f^{(3)} to f^{(5)}, 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 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 the kth contact has just moved from C to NC. When f was still in C, we computed Δ f_i by solving the system A_{11} x = –v_i and setting Δ f_i = x_i. Let A_{11} and x be partitioned by

\[ A_{11} = \begin{pmatrix} B & w \\ \beta & \end{pmatrix}, \quad x = \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} z \\ c \end{pmatrix} = -v_i \] (31)

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

\[ u = B^{-1}(z - wy) \]

or

\[ w^T B^{-1}(z - wy) = c - \beta y \] (32)

Since this Δ f caused f_k to decrease to zero, Δ f_k = y must have been negative.

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

\[ B \tilde{u} + w \tilde{y} = z \] (34)

to obtain

\[ \tilde{u} = B^{-1}z. \] (35)

From equations (8) and (31), the new Δ a_k is

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

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

\[ \Delta a_k = w^T B^{-1}z - c = -w^T B^{-1}(wy - \beta y) = -y(w^T B^{-1}w + \beta). \]

Since A_{11} is positive definite, B^{-1} is positive definite, and \beta is positive, so w^T B^{-1}w + \beta must be positive. Since y is negative, –y is positive, and we conclude that Δ 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 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**


Figure 1: Time-lapse simulation sequence of a blockfeeder.
Figure 2: Time-lapse simulation sequence of a double-action jack.