Interactive Manipulation of Articulated Objects with Geometry Awareness

Min-Hyung Choi
choi@cs.uiowa.edu

James F. Cremer
cremer@cs.uiowa.edu

Computer Science Department
The University of Iowa
Iowa City, IA 52242

Abstract

Techniques for interactive 3D manipulation of articulated objects in cluttered environments should be geometrically aware*, going beyond basic inverse or forward kinematics to allow contact while preventing interpenetration. This paper describes a general purpose interactive object manipulation technique using nonlinear optimization. The method converts geometry awareness into sets of inequality constraints and handles nonlinear equality and inequality constraints efficiently without restricting object topology. Our iterative algorithm has a quadratic convergence rate and each iteration can be solved in $O(n_{nz}(L))$, where $n_{nz}(L)$ is the number of non-zeros in L, a Cholesky factor of a sparse matrix. To promote additional speedup, symbolic factorization is separated from numerical computation. Our approach provides a framework for using optimization techniques in interactive tools for building and manipulating models in constrained, cluttered environments.*

1 Introduction

Many interactive graphical applications incorporate direct manipulation as a dominant interaction method for configuring and arranging graphical objects. Emerging applications in 3D virtual environments, such as scene setup, object assembly, and posing a virtual character, require more sophisticated yet natural manipulation schemes. Traditional manipulation techniques that employ forward kinematics or other direct manipulation of object state do not provide convenient control of complex object configurations. Techniques based on inverse kinematic and optimization techniques address this problem, but so far are limited to specific structures or are too slow for interactive use.

The most serious deficiency of previous approaches is that they fail to support collision detection and penetration avoidance, which ensure physically plausible object behavior during manipulation. This lack of *geometry awareness* can be crippling in cluttered environments where contact and interaction between objects is natural and desirable. A system that automatically prevents interpenetration during manipulation can make complex scene creation and character manipulation substantially easier and less tedious.

An interactive graphical manipulation system should meet several requirements. First, it must be fast enough to be used interactively regardless of both object configuration topology and constraint types. Object and constraint representations should be modular and flexible so that new types of constraints can be easily added. Under- and over-determined situations should produce reasonable behavior. And, geometry awareness should be provided so that object behavior during manipulation is plausible.

This paper presents a method for geometrically-aware interactive manipulation of 3D objects, emphasizing the numerical algorithms. In our approach, we cast object manipulation into a nonlinear optimization problem involving equality and inequality constraints. The method does not restrict object topology, and uses unreduced coordinates for flexibility and scalability. Interactive speed results from taking advantage of the special structure and sparsity of the problem. Section 2 presents background material and related work. In section 3, we give an overview of our interactive system. Section 4 presents the formulation of the nonlinear optimization problem used for object manipulation. Section 5 details the method for efficiently solving the optimization problem.

2 Background and Related Work

The importance of sophisticated 3D manipulation techniques is based partly on a desire for interaction that looks and feels similar to real world object manipulation. To provide a consistent and intuitive interaction that resembles interaction with real world objects, we provide users a simple metaphor: grab any part of the object and drag or pull it to the intended configuration, letting the system take care of the underlying computation.

The inverse kinematics problem has been studied extensively in the robotics field, although it is fairly recently that the techniques have been adopted in computer graphics and animation. In robotics, much of the effort is concerned with the functionality of particular manipulators, sometimes with a few redundant degrees of freedom. Extension of robotics techniques to more general configurations and situations will be of great value in computer graphics and virtual environments applications.

In computer graphics, there have been a number of attempts to address the problem of manipulation of redundant articulated objects. Jack[5] uses a reduced coordinate system and nonlinear optimization techniques to solve inverse kinematics for a human model. Zhao[17] added collision avoidance to Jack but this simple augmentation using bounding spheres and cylinders does not support precise geometry awareness to allow contact or prevent interpenetration between objects in close proximity. Gleicher's[11] differential method casts the graphical manipulation problem to a nonlinear optimization problem, linearizing it to make the problem easier to solve.

Snyder[13] addresses geometry awareness but not for articulated objects. The work investigated the generation of stable non-interpenetrating configurations of curved-surface objects.

Interactive dynamics simulation may also be used as a basis for object manipulation[6, 15]. Interactive dynamics simulation can provide fairly physically accurate results, accounting for friction and inertia in addition to geometry awareness. As good haptic feedback interfaces are developed, advanced interactive dynamic approaches should prove quite attractive. The reason for our investigation of a purely kinematic approach is that we feel that the use of dynamics raises as many problems as it addresses. The additional quantities involved in the computations contribute to generally less efficient and more complicated algorithms. Furthermore, an advantage of a kinematics approach is that intermediate states in a manipulation do not need to be dynamically plausible. Our primary goal is to give users control over the manipulation and to get reasonable results quickly.

Figure 1: A challenging manipulation problem

Figure 1 shows initial and final configurations of a challenging problem in human figure manipulation: sitting on a chair. A good manipulation system would allow users to interactively drag the human figure from a standing to a sitting position, automatically allowing contact but preventing penetration between the human and the chair.

3 Overview of approach

We have developed an interactive 3D object manipulation system incorporating the algorithms described in this paper. Joints or hinges are modeled with equality constraints while "geometry awareness" is achieved by using inequality constraints to prevent object interpenetration. The system supports three interaction techniques, each appropriate in different situations:

For *Dragging*, the system continuously polls the mouse location and uses the polled locations as intermediate goals. In this mode, we require the algorithm to quickly achieve a moderately accurate solution at each intermediate goal, with a more accurate solution at the end of move. For smooth interaction, the system must provide intermediate goal solutions within the interval between mouse polls (typically, approximately 0.1 second).

Pull to goal may be used to specify goal trajectories. The user specifies a desired path for points on the manipulated object. Based on the path, the system generates closely spaced intermediate goals and solves for them, animating the intermediate results.

In the *Solve for goal* scheme, a user directly specifies desired final goal constraints. The system solves for and displays a configuration meeting the constraints.

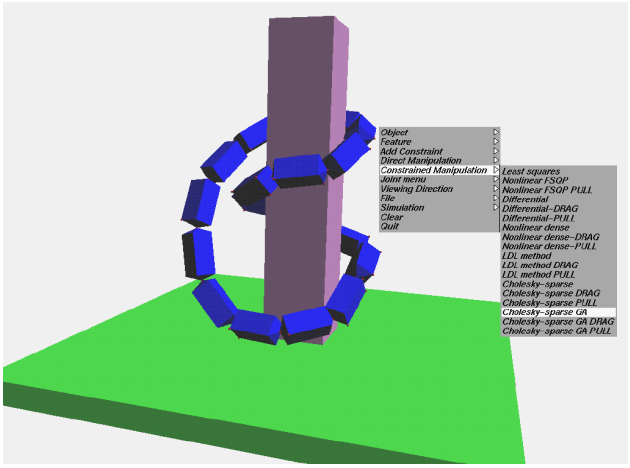

Figure 2: Snapshot of manipulation interface

Objects jump to the new configuration, providing the user no information about how to reach it from the initial configuration.

The different interaction schemes involve solving nonlinear optimization problems using different step sizes and varying distances between intermediate goals. "Constraint drift" is not a problem since constraints are satisfied within default or user-specified tolerances at each intermediate goal, as well as at the final goal. The following is an outline of the top-level implementation of the *Dragging* interaction scheme.

```
while (manipulation is not finished) {
    Generate intermediate goal by polling mouse;
    Save current state;
    if(constraint update conditions are met)
      Update active constraints set;
    Solve one step of nonlinear optimization;
    if(interpenetration has occurred)
     {
      Restore previous state;
      Using interpenetration and nearest distance
        information, formulate inequality constraints
        that will prevent interpenetration in next step;
      Add inequality constraints to optimization problem;
      Solve one step of nonlinear optimization;
     }
}
```

Figure 2 is a snapshot from the user-interface of our system handling a closed-loop articulated objects. The system allows users to assemble complex figures by adding and deleting constraints interactively. The software is modular and supports comparison of a variety of object manipulation algorithms. In addition to the new methods described in this paper, the software includes implementations of several other methods.

4 Formulation: Object Manipulation as Nonlinear Optimization

We cast object manipulation as a constrained nonlinear optimization problem. Considering that there is an infinite number of solutions for the underdetermined case and we want to get a reasonably close solution when an exact solution is not available, nonlinear optimization has a number of advantages. Depending on the situation, we can experiment with different objective functions while still maintaining kinematic constraints. Furthermore, incorporation of inequality constraints is not very difficult in our optimization approach. Equation 1 represents an objective function to achieve the "Principle of Least Astonishment"[7] which suggests the system should move as little as possible from the previous configuration.

$$E(q) = \frac{1}{2}\left\|q - q_0\right\|^2 \tag{1}$$

Combining this objective function $E(q)$ with kinematic equality constraints, $\Phi_i(q)$, and inequality constraints, $\Phi_j(q)$ for geometric awareness, we formulate the object manipulation problem as the nonlinear optimization problem:

$$\begin{array}{lll} minimize & E(q) & q = q_1, q_2, .., q_n \\ s.t. & \Phi_i(q) = 0, & i = 1..m_{eq}, j = 1..m_{ineq} \\ & \Phi_j(q) > 0, & m_{eq} + m_{ineq} = m \end{array} \tag{2}$$

4.1 Object representation

Object representations fall into two general categories, reduced coordinates and unreduced, or full, coordinates[6]. Reduced coordinate approaches describe objects using minimal sets of state variables, corresponding in size to the number of available degrees of freedom. With reduced coordinate approaches, constraint drift does not occur, but it is relatively difficult to modify the constraint set on-line and efficiently reparameterize it to new minimal coordinates. In contrast, full-coordinate methods express system state using a maximal set of coordinates corresponding to degrees of freedom of objects when unconstrained. Constraint maintenance is achieved by explicitly accounting for constraints in the solution methods.

We describe each object's state with 7 variables: r_x, r_y, r_z for the center of the object's local reference frame, and Euler parameters e_r, e_x, e_y, e_z for the object's orientation. Employing full coordinates and explicit constraints facilitates development of a flexible

algorithm that supports on-line constraint set changes and that can be implemented cleanly and modularly. Many useful constraints, including but not limited to standard mechanical joints, are conveniently represented by nonlinear equalities. Examples include:

- spherical joint : $r_1 + A_1c_1 - r_2 - A_2c_2 = 0$, where A_i is a rotation matrix dependent on the Euler parameters and c_i is a body-fixed vector describing the joint attachment point.
- point - surface attachment: $n_2 \cdot ((r_1 + A_1c_1) - p_2) = 0$, where n_2 is the surface normal and p_2 a point on the surface.
- point goal: $r_1 + A_1c_1 - p_{goal} = 0$, where p_{goal} is the goal position for the specified object 1 point.

4.2 Geometry awareness as nonlinear inequality constraints

To incorporate geometry awareness into the manipulation system, robust and very efficient collision detection and nearest feature identification methods must be tightly integrated with the nonlinear optimization algorithms. Most collision detection packages do not meet the needs of manipulation systems; some provide simple "interpenetrating/not interpenetrating" results, while others compute minimum distances without robustly identifying nearest features. We use a new geometric computing package developed by Mirtich[12] that provides very fast collision detection, distance computation, and nearest feature identification.

In our manipulation algorithm, interpenetration checks are performed at every iteration. When interpenetration occurs, the state at the previous step is restored, and nearest feature information is used to formulate a non-interpenetration inequality constraint. This inequality constraint is added to the optimization problem's constraint set, preventing reinterpenetration of the offending features when the solution step is retried.

The size of the inequality constraint set is kept small by removing inequalities when feature distance grows beyond a certain size or a feature leaves the relevant Voronoi bounding region. In our approach, geometric analysis is somewhat different from existing methods in dynamic simulation because constraints are monitored at non-very-small distances. Formulating and maintaining the appropriate inequality constraint set is an interesting and non-trivial problem. Details of this aspect of geometric analysis are presented in [9]. In this paper, assuming the appropriate geometric inequality constraints are given, we focus on the algorithms for efficiently solving the formulated optimization problem.

5 Solving the Nonlinear Optimization Problem

The key challenge for achieving interactivity is efficiently solving the constrained nonlinear optimization problem. Our method converts the optimization problem into a system of nonlinear equations. We can exploit special aspects of the formulation to achieve the needed performance. First, the system is sparse and block structured. Second, each solution step begins from a starting point at which all constraints except the goals are satisfied.

Assuming that inequality constraints have been replaced by equalities as described in Section 5.5, the Lagrangian function[10] is defined as

$$L(q, \lambda) = E(q) + \sum \lambda_i \Phi_i(q). \tag{3}$$

The first order necessary condition for the existence of a local minimizer $q*$ of the objective function of Equation 1 requires the existence of a Lagrange multiplier λ^* satisfying Karush-Kuhn-Tucker(KKT) condition

$$\nabla q L(q^*, \lambda^*) = \nabla E(q^*) + \sum \lambda_i \nabla \Phi_i(q^*) = 0 \tag{4}$$

Combining this with the original constraints yields

$$F \begin{pmatrix} q \\ \lambda \end{pmatrix} = \begin{Bmatrix} q - q_o - J(q)^T \cdot \lambda = 0 \\ \Phi_i(q) = 0 \end{Bmatrix} \tag{5}$$

where J is the Jacobian of the constraints Φ_i. The nonlinear optimization problem has become a well-balanced system of $n+m$ equations in $n+m$ variables. Newton's method converges quadratically if the starting point is within a certain range[10]. Depending on the choice of user interface scheme, and the distance between intermediate goals, the starting point may not be in the range that guarantees convergence. If the intermediate goal is too far away for the numerical system to converge, the path is interpolated and subdivided to provide a sequence of intermediate goals. We can solve the nonlinear system as follows:

$$\nabla F = \begin{bmatrix} I & -J^T \\ -J & 0 \end{bmatrix} \tag{6}$$

$$\nabla F \delta = -F \begin{pmatrix} q \\ \lambda \end{pmatrix}, \quad q_{n+1} = q_n + \delta \tag{7}$$

Since *Dragging* produces very close goals, and correspondingly good starting points, less than 3 iterations of system solving usually suffice. The iteration is terminated if the errors for all equations are less than 1.0E-5. In graphical object manipulation where objects are displayed on a rasterized screen, a tolerance within one pixel is acceptable in many applications.

5.1 Two possible solution techniques

The key issue is to solve linear system 7 efficiently. If we rewrite equation 7 as

$$\begin{bmatrix} I & -J^T \\ -J & 0 \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{8}$$

and multiply the upper row by J , then

$$\begin{aligned} J\delta_1 - JJ^T\delta_2 &= Jb_1 \\ -J\delta_1 &= b_2 \end{aligned} \tag{9}$$

leading to

$$JJ^T\delta_2 = -b_2 - Jb_1 \tag{10}$$

We call equation 10 a normal equation of the linear system 8. This normal-equation-based method has a number of desirable properties. First, assuming that J has full rank, $JJ^T$is symmetric and positive definite. Furthermore, the size of the system is usually significantly smaller than the original linear system 7. One drawback is that even if J is sparse, JJ^Tcan be completely dense depending on the topology of the bodies. For example, a star topology, in which every object is constrained with a central object, has completely dense JJ^Teven when J is sparse, due to the existence of single dense column.

The other solving technique is to solve 8 directly using LDL^T factorization, because it is always sparse regardless of the structure of J. Vandervei [14] showed that any symmetric permutation of a quasidefinite matrix has a factorization LDL^T and proposed an efficient sparse algorithm. The main advantage of this approach is that it is not hampered by a few dense columns in J.

In this paper, we show how to alleviate the problem of dense columns in the normal equation approach. When dense columns occur, we rearrange and expand the system. When there are no dense columns we we simply take advantage of small system size of the normal equation and use efficient LL^T Cholesky factorization. Studies show that the Cholesky factorization based on normal equation method is usually preferred [16] if the upper left corner block of the linear system 8 is a diagonal matrix.

5.2 The dense columns in JJ^T

To avoid dense decomposition we employ the modified Schur complement method[4]. The main idea is to rewrite the normal equation system 10 into

$$\begin{bmatrix} (A_sA_s^T + FF^T) & A_D & F \\ A_D^T & -I & 0 \\ F^T & 0 & I \end{bmatrix} \begin{bmatrix} \delta \\ \gamma \\ \rho \end{bmatrix} = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix} \tag{11}$$

where A_s and A_D are sparse and dense columns of J respectively. Since we separate J, there's no guarantee that $A_sA_s^T$ is nonsingular[4]. We choose F such that each column of F contains only one non-zero to make

$$A_sA_s^T + FF^T \tag{12}$$

nonsingular. This demonstrates that FF^T only modifies the diagonal elements of $A_sA_s^T$ and the revised matrix can be solved with the same sparse Cholesky factorization as the original matrix.

5.3 Separating Symbolic Factorization from Numerical Computation

To take advantage of sparsity and block structure, we separate symbolic factorization from numerical computation. First, row and column orderings are determined such that the fill-in in the Cholesky decomposition L is minimized. Since the problem of choosing an ordering that minimizes fill-in is NP-hard[2], we use the approximate minimum degree heuristic(AMD)[1]. Second, the non-zero structure of the Cholesky factor is determined and sparse storage is allocated accordingly. This takes care of indexing and associated bookkeeping so that the actual numerical factorization takes place only for the non-zero elements. If the structure of $JJ^T$is unchanged during the iterations, we can do the symbolic decomposition just once, repeating only the numerical part thereafter.

5.4 The Push Cholesky Algorithm

Our sparse factorization method is based on the push Cholesky algorithm[2]. $L_{(i:j)k}$ denotes the kth column from row i to j.

1. for j = 1, ... m
2. $L_{jj} := \sqrt{L_{jj}}$
3. $L_{(j+1:m)j} := L_{(j+1:m)}/L_{jj}$
4. for k = j + 1, ... m
5. if $L_{kj} \neq 0$
6. $L_{(k+1:m)k} := L_{(k+1:m)k} - L_{kj}L_{(k+1:m)j}$

THE PUSH CHOLESKY ALGORITHM

Although the algorithm is presented in columns, the actual computation is usually performed for few elements in a column. For example, in step 6, only the non-zero components of $L_{(k+1:m)j}$ are subtracted. However, since a column is used to update another column, when the non-zero structures of two columns, $L_{.j}$ and $L_{.k}$, don't correspond, we pay a bookkeeping penalty in the sparse algorithm. A supernode technique is used to get around this problem and eventually to achieve faster sparse solving [2]. A supernode is a set of adjacent columns in L with the same non-zero pattern below the diagonal of the previous column[2]. By regrouping columns using the supernode technique, all columns in the supernode update exactly the same position in the subsequent columns. Furthermore, loop unrolling is possible by employing the supernode[3]. With most common memory architectures having at least 2 levels of cache, loop unrolling leads to speedup by taking advantage of locality of memory access. Since the loop from step 4 to step 6 is performed only for few non-zero elements in a column, the asymptotic complexity of this sparse Cholesky factorization is $O(n_{nz}(L))$[2], where $n_{nz}(L)$ is the number of non-zeros in L.

5.5 Inequality Constraints

A common way to handle inequality constraints in nonlinear optimization is to add a logarithmic barrier function. As in 13, we take the logarithm of all inequality constraints and subtract the resulting terms from the original objective function.

$$\begin{array}{c} minimize\ \|q-q_0\|^2 - \mu\sum \log \Phi_j(q) \\ s.t.\ \ \Phi_i(q) = 0 \\ i = 1..m_{eq},\ j = 1..m_{ineq},\ \ m_{eq} + m_{ineq} = m \end{array} \tag{13}$$

If any inequality constraint is violated, the log function goes to negative infinity and, consequently, objective function goes to the positive infinity. For the Newton's direction, we need ∇F and the linear system becomes

$$\begin{array}{c} \left[\begin{array}{cc} W & -J^T \\ -J & 0 \end{array}\right]\left[\begin{array}{c} \delta_1 \\ \delta_2 \end{array}\right] = \left[\begin{array}{c} b_1 \\ b_2 \end{array}\right] \\ W = \nabla^2 E(q) + \mu\sum\left[\frac{1}{\Phi_j^2(q)}\nabla\Phi_j(q)\nabla\Phi_j(q)^T - \frac{1}{\Phi_j(q)}\nabla^2\Phi_j(q)\right] \end{array} \tag{14}$$

where W represents a derivative of the objective function. An implicit assumption is that the starting point must satisfy all inequality constraints. There are two main disadvantages of this approach. An appropriate coefficient for the logarithmic barrier function must be selected to ensure convergence and numerical stability. Furthermore, choosing an efficient sparse linear system solving algorithm may depend on the structure of the upper left corner block W. The normal equation method requires an inverse of W. Since W is a general symmetric matrix, it is costly to invert. W becomes diagonal matrix only if all the inequality constraints are simple bounds. A modified nonlinear optimization formulation 15 shows how to eliminate problems related to the barrier function approach.

$$\begin{array}{cc} minimize & E(q,Y) \\ s.t. & \Phi_i(q) = 0 \\ & \Phi_j(q) - Y_j^2 = 0 \end{array} \tag{15}$$

All inequality constraints are converted into equality constraints by introducing new variables Y. The square of Y ensures that $\Phi_j(q)$ is greater than or equal to zero. Since we don't use logarithmic barrier function, the upper left block W becomes an identity matrix. Therefore, the push Cholesky algorithm based on the normal equation formulated for systems with equality constraints can be used directly. The number of variables increases by the number of active inequality constraints.

To ensure efficiency, it is important to keep the set of active inequality constraints as small as possible. As mentioned in Section 4.2, details of our geometric analysis are presented in [9][8]. The monitoring and management of inequalities are important for the overall performance, because constraint addition or deletion can alter the block structure and necessitate further symbolic manipulation.

5.6 Over-determined case

Conflicting or unreachable goals yield a singular constraint matrices. Singular value decomposition methods can be used to solve such problems, but are expensive. In some cases, we alter the objective function to help the system find a reasonably close solution in a least-squares sense. The objective function then becomes a least squares summation of the Euclidean distance between goal positions. In other cases, we use a different technique, called *damping* to make the matrices non-singular. By altering JJ^T with small diagonal elements of μI, we can avoid trying to solve a singular system.

From a user interface point of view, over-determined cases can be avoided or solved by providing users a chance to evaluate the situation. Since users get continuous feedback, when there are conflicting goals or the goals can't be reached, users might

know how to guide the the system around the problem. Similar user interface scheme can be applied when the system is trapped in local minima.

6 Results and concluding remarks

In this paper, we described a general purpose "geometry aware" 3D object manipulation technique based on constrained nonlinear optimization. Overall, this work combines new developments in geometric analysis and nonlinear optimization to provide interactive manipulation capability beyond what has been provided before. The efficient sparse algorithm allows users to manipulate complex object configurations interactively. The method scales well with the number of objects and active equality and inequality constraints. Our experiments show that the push Cholesky algorithm performs well and scales nearly linearly. Tests were run on an SGI O2 workstation with a 175MHz R10000 processor and a PC with 400MHz Pentium II processor. A test configuration with 212 variables and 1455 non-zero elements, roughly 96% sparse, takes less than 7.5ms per solution on the PentiumII. This involves three sparse Cholesky solves and function evaluations. System performance does not degrade in the presence of closed loop configurations or inequality constraints.

Although various interaction metaphors are available in the form of 3D widgets, it remains difficult to manipulate 3D figures with a 2D input device. Further work is necessary to develop effective 3D interfaces supporting powerful yet convenient interactions. Extension of this work to include deformable objects is another area ripe for future work; the human figure manipulation "challenge problem" of Figure 1, cannot be solved fully adequately without accounting for object deformability.

Acknowledgments Partial support for this work was provided by Office of Naval Research grant N00014-96-1-0269. We thank Yinyu Ye and Xiong Zhang for helpful advice on the formulation and solution of optimization problems.

References

[1] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, October 1996.

[2] E. D. Andersen and K. D. Andersen. The APOS linear programming solver: an implementation of the homogeneous algorithm. Technical report, Department of Management, Odense University, Denmark, March 1995.

[3] E. D. Andersen, J. Gondzio, C. Mészáros, and X. Xu. Implementation of interior-point methods for large scale linear programs. In *Interior point methods of mathematical programming*, pages 189–252. Kluwer, Dordrecht, The Netherlands, 1996.

[4] Knud D. Andersen. A modified Schur-complement method for handling dense columns in interior-point methods for linear programming. *ACM Transactions on Mathematical Software*, 22(3):348–356, 1996.

[5] Norman I. Badler, Cary B. Phillips, and Bonnie Lynn Webber. *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press, New York, 1993. ISBN 0-19-507359-2.

[6] D. Baraff. Linear-time dynamics using Lagrange multipliers. *Computer Graphics (SIGGRAPH '96 Proceedings)*, 30(2):137–146, 1996.

[7] Lee Alton Barford. A graphical, language-based editor for generic solid models represented by constraints. Technical Report TR87-813, Cornell University, Computer Science Department, March 1987.

[8] Min-Hyung Choi. *Geometry Awareness for Interactive Object Manipulation*. Ph. D. thesis, Department of Computer Science, University of Iowa, 1999.

[9] Min-Hyung Choi and James F. Cremer. Geometric awareness for interactive object manipulation. In *Proceedings of Graphics Interface '99*, June 1999.

[10] R. Fletcher. *Practical Methods of Optimization,* 2nd ed. John Wiley, New York, 1987.

[11] M. Gleicher. *Differential Approach to Graphical Manipulation*. Ph. D. thesis, School of Computer Science, Carnegie Mellon University, 1994.

[12] Brian Mirtich. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[13] J. Snyder. An interactive tool for placing curved surfaces without interpenetration. *Computer Graphics (SIGGRAPH '95 Proceedings)*, 29(2):209–218, 1995.

[14] R. Vanderbei. Symmetric quasi-definite matrices. *SIAM Journal of Optimization*, 5(1):100–113, 1995.

[15] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. In *Symposium on Interactive 3D Graphics*, pages 11–21, March 1990.

[16] Y. Ye and E. D. Andersen. On a homogeneous algorithm for the monotone complementarity problem. Technical report, Department of Management Sciences, The University of Iowa, 1997.

[17] Xinmin Zhao and Norman Badler. Near real-time body awareness. *Computer Aided Design*, 26(2):248–253, 1994.