Geometric Awareness for Interactive Object Manipulation

Min-Hyung Choi   James F. Cremer

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

Abstract

This paper describes formulation and management of constraints that, combined with a nonlinear optimization algorithm, enable interactive geometrically aware manipulation of articulated objects. Going beyond purely kinematic or dynamic approaches, our solution method directly employs geometric constraints to ensure non-interpenetration during object manipulation. We present the formulation of the inequality constraints used to ensure nonpenetration, describe how to manage the set of active inequality constraints as objects move, and show how these results are combined with a nonlinear optimization algorithm to achieve interactive geometrically aware object manipulation. Our optimization algorithm handles equality and inequality constraints and does not restrict object topology. It is an efficient iterative algorithm, quadratically convergent, with each iteration bounded by $O(n_{nz}(L))$, where $n_{nz}(L)$ is the number of non-zeros in L, a Cholesky factor of a sparse matrix.

Key words: object manipulation, geometry awareness, constraints, nonlinear optimization, Cholesky

1   Introduction

Interactive object manipulation is typically achieved in computer graphics applications via direct manipulation of degrees of freedom or inverse kinematics algorithms. The importance of efficient, convenient, and natural manipulation techniques is growing. As 3D virtual environment applications mature, the creation and arrangement of realistic scenes cluttered with large numbers of objects, many of them in contact, remains a significant challenge.

A serious deficiency of many object manipulation techniques is that they are not *geometrically aware*. That is, although they maintain kinematic constraints, they do not enforce the basic geometric constraint that objects should not interpenetrate. A geometrically aware object manipulation system enforces both kinematic and geometric constraints, allowing manipulation in which objects may come into contact but do not interpenetrate. A lack of geometry awareness can be crippling in cluttered environments where contact and interaction between objects is natural and desirable.

An effective object manipulation system should satisfy several requirements. First, it must be fast enough to be used interactively for general object configuration topologies and a variety of constraint types. It should include geometry awareness so that object behavior during manipulation is plausible. The system should be extensible to new object and constraint types. And, the underlying manipulation algorithms should produce reasonable behavior in under- and over-determined situations.

This paper presents our approach to geometrically aware interactive object manipulation. We formulate and manage a set of inequality constraints to prevent interpenetration during manipulation. The manipulation problem is cast into a nonlinear optimization problem involving equality and inequality constraints. Our solution 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. Constraint management keeps the number of active constraints small to reduce system size and enhance overall efficiency. Section 2 presents background material and related work. Section 3 presents the formulation of the inequality constraints providing geometry awareness, and discusses management of the constraint set. In section 4, we describe our interactive object manipulation system, and the formulation of the nonlinear optimization problem that lies at the heart of the manipulation algorithm. 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 manipulation. To provide intuitive interaction that resembles interaction with real world objects, we provide 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 ex-

Figure 1: A challenging manipulation problem

tensively in the robotics field, but much of the effort is concerned with the functionality of particular manipulators, sometimes with a few redundant degrees of freedom. In computer graphics, there have been a number of attempts to address the problem of manipulation of redundant articulated objects. Jack[4] uses a reduced coordinate system and nonlinear optimization techniques to solve inverse kinematics for a human model. Zhao[16] 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[9] 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[11, 5, 14]. 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 dynamics approaches should prove quite attractive. We chose to investigate a purely kinematic approach because at least for some applications, dynamics is unwarranted and raises as many problems as it addresses. For instance, the additional quantities (e.g., force and inertia) involved lead to generally more complex and less efficient 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 more manipulation control and to get reasonable results quickly.

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 Geometry Awareness

To incorporate geometry awareness into the manipulation system, robust and very efficient collision detection and nearest feature identification methods must be tightly integrated with our nonlinear optimization algorithm. 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 system developed by Mirtich[12] that provides very fast collision detection, distance computation, and nearest feature identification. The system includes hierarchical bounding volume methods similar in style to those used in OBBTree[10], combined with hierarchy optimization techniques, and efficient nearest distance witness identification algorithms.

Assuming that the geometry system provides, at each iteration, either a report of interpenetration, or nearest feature witnesses, geometry awareness is achieved through *formulation* of appropriate inequality constraints to prevent interpenetration, and *management* of a set of currently active constraints.

3.1 Constraint Formulation

At each step in which there is no interpenetration, the geometry system computes and reports a nearest features witness [1] for pairs of objects that are in close proximity. A nearest features witness is one of six types: (1) vertex-face, (2) vertex-edge, (3) vertex-vertex, (4) edge-edge, (5) edge-face, and (6) face-face. As detailed in Mirtich[12], if $F1$ and $F2$ are features (vertex, edge, or face) from disjoint convex polyhedra, and points $p1$ on $F1$ and $p2$ on $F2$ are the closest points between $F1$ and $F2$, then $F1$ and $F2$ represent the closest features witness *if* $p1$ lies in the Voronoi region of $F2$ and $p2$ lies in the Voronoi region of $F1$. [2]

Given a nearest features witness, our goal is to prevent subsequent feature interpenetration by formulating an ap-

---

[1] For convex polyhedra, only one nearest features witness is needed. Our approach also works for nonconvex polyhedra, as long as the geometry system can report multiple near rather than nearest features , but we limit the presentation here to convex objects.

[2] For a feature F on a convex polyhedron, the Voronoi *region* is the set of points outside the polyhedron that are as close to F as to any other feature on the polyhedron. Voronoi *planes* are the boundary planes that separate neighboring Voronoi regions.

propriate inequality constraint to include in the nonlinear optimization problem we solve during manipulation. We consider each case in turn:
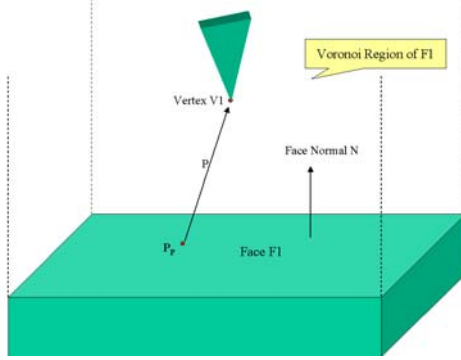
Vertex-face

Figure 2: Vertex-Face condition

The vertex-face case is simple; a point-plane inequality constraint is sufficient to prevent interpenetration. Figure 2 shows a vertex is inside of a Voronoi region of a face. The non-penetration inequality is

$$N \cdot P > 0, \qquad (1)$$

where $P = V1 - P_p$ and P_p is some point on the plane of the face. This constraint is quadratic because N and P are actually both functions of object position and orientation.

Vertex-edge

For the vertex-edge case, a separating plane is used to prevent interpenetration. We choose a plane that contains the witness edge and has normal equal to the average of the normal vectors of faces adjacent to the edge. Figure 3 illustrates the vertex-edge case from two different point of view. The separating plane does not allow the vertex to move to every penetration-free position, and may thus appear to overly constrain movement. It is important to realize that this is *not* the case. Before reaching the boundary of the constraint region, the nearest feature witness must change from the vertex-edge to some other case. For instance, if the vertex crosses a Voronoi plane on the way toward the separating plane, the geometry system will detect a change in closest feature witnesses and the constraint set will be updated appropriately. Note also there are many valid separating planes. A plane is valid if it contains the edge and positive dot products exist between its normal and the normals of both Voronoi bounding planes of the edge.

Thus, as in the vertex-face case, the vertex-edge case is handled with an inequality constraint of the form $N \cdot P > 0$.

Vertex-vertex

Similar to the vertex-edge case, a separating plane through one vertex, v, is used to prevent interpenetration as shown in Figure 4. Again, note that many valid separating planes exist. A valid separating plane must simply have a normal, n, such that a ray (v, n), originating at the vertex and having direction equal to the normal vector, must lie completely in the vertex Voronoi region. We can simply choose a separating plane normal by averaging the normals of the faces adjacent to the vertex. In certain unusual cases, this averaging can yield a ray (v, n) that lies close to the vertex Voronoi region boundary. It will always be inside, however, and thus valid.

For this case we again formulate a point-plane inequality constraint: $N \cdot P > 0$. And again, while the separating plane does not permit direct movement into some regions, it permits full movement *to the extent required* by the vertex-vertex case. When the witness changes the constraints will change appropriately to permit movement into other regions.

Edge-edge

For the edge-edge case, constraints are formulated in terms of intersection points between one edge and extended planes of the adjacent faces of the other edge. Figure 5 depicts an edge-edge case, involving egdes E_A and E_B. Let $EP1$ and $EP2$ be extended planes of the adjacent faces of E_B. Edge-edge penetration occurs when E_A intersects the faces adjacent to E_B. Thus, to prevent intersection we can simply prevent *one* of those situations. We formulate another separating plane constraint that requires that the intersection of the extended line EL containing E_A and, say, $EP1$, occurs on the same side of the separating plane as E_B's Voronoi region. The separating plane is chosen exactly as in the vertex-edge case. Note that it is possible for E_A to penetrate the other object while satisfying the constraint. However, this situation will not be reached in the edge-edge case because a change in nearest witness type will occur before reaching this situation.

So, the edge-edge case is handled by another point-face inequality constraint. Here, the constraint is slightly more complicated since the point is not a specific point on one object, but is defined as the intersection point between EL and $EP1$. This point changes as objects move and therefore is is represented analytically in the equation rather than being explicitly computed at iterations of the algorithm. The form of the constraint is

$$(EL \ intersection \ EP1) \ outside \ of \ SP$$

Face-face and Edge-face

Edge-face and face-face witnesses occur only when the nearest features are an edge or face parallel to a face of
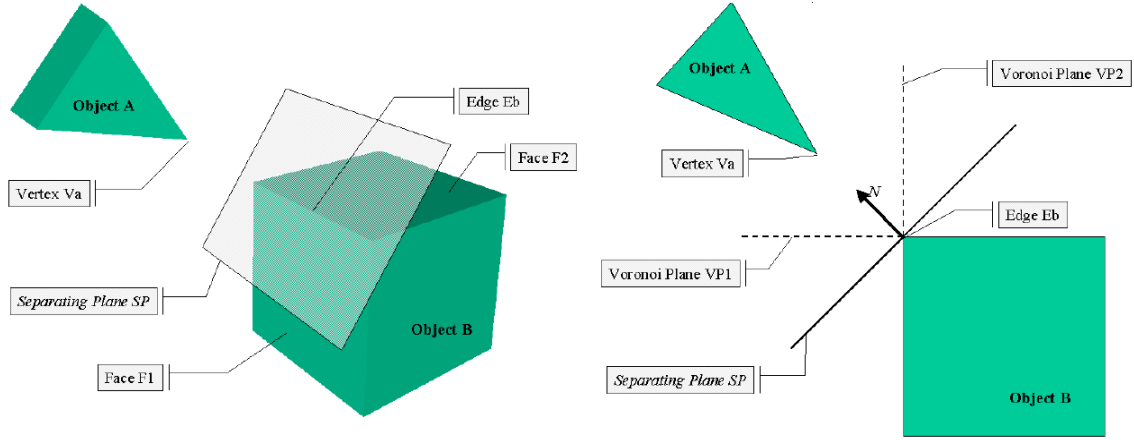
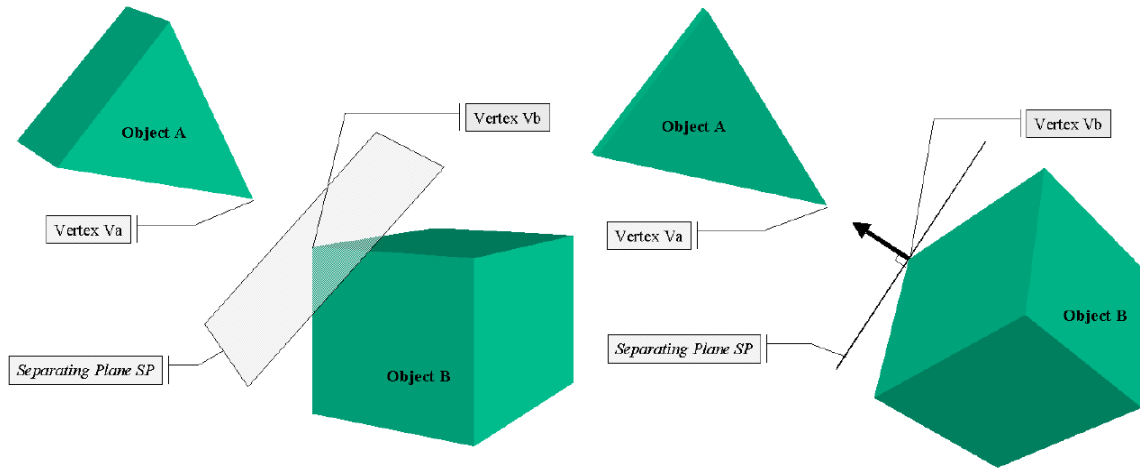Figure 3: Vertex-Edge condition

Figure 4: Vertex-Vertex condition

the other object. It is degenerately rare when the nearest distance is not zero, but routine when objects are in contact. Both cases are handled by reducing them to a set of vertex-face and/or edge-edge witnesses. For example, in the edge-face case, if both edge vertices are inside the Voronoi region of the face, two vertex-face witnesses are used. If the edge penetrates one of the Voronoi boundary planes of the face, the case is reduced to one edge-edge case and one vertex-face case.

3.2 Management of Active Constraints

As objects move and nearest features witnesses change, our system adds and removes inequality constraints to reflect the current situation. To ensure efficiency, it is important to keep the set of active inequality constraints as small as possible.

Geometry queries are done at each iteration of the algorithm to determine nearest features or detect interpenetration. Two approaches may be used for deciding when to add inequality constraints corresponding to near features. The system can wait until interpenetration occurs, revert to the state from the previous iteration, and formulate and add to the constraint set the appropriate inequalities based on nearest features. The inequalities will prevent re-interpenetration of the offending features when the solution step is retried. Alternatively, the system can preventatively add inequalities once nearest feature distances fall below some threshold. Our system employs both approaches; it preventatively adds constraints when features get close, but also includes the backup-upon-interpenetration technique. (This can enhance robustness when we employ large step sizes in iterations of manipu-
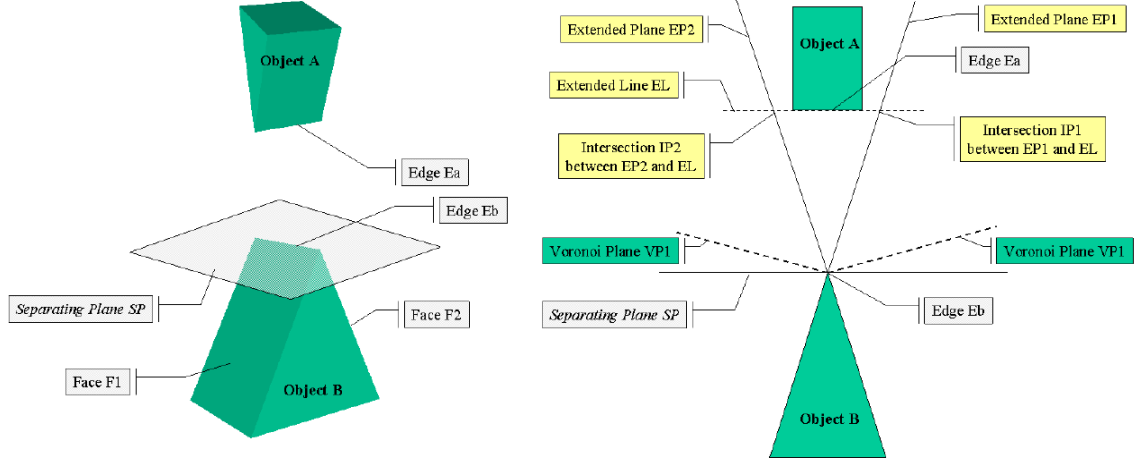
Figure 5: Edge-edge condition

lation algorithm during, for instance, debugging or other situations where we desire especially fast but perhaps less precise performance).

Inequalities are removed from the active constraint set under two conditions, one representing a required removal, the other an optional removal. Required removal occurs when nearest feature witnesses change; the system removes the corresponding inequality already in the constraint set and adds a new one as indicated above. Inequalities may be optionally removed, to keep the constraint set small, when the distance corresponding to a nearest feature witness grows beyond some threshold.

Maintaining a minimal number of active constraints at all times results in small, more efficiently solvable constraint sets for the optimization algorithm. On the other hand, frequent modification of active constraint set can adversely affect overall performance. Our solution method incorporates a separation of symbolic and numerical computation components to enhance performance when several continuous steps are taken without changing the system structure. Constraint addition and deletion can alter the block structure of the system and necessitate further symbolic manipulation, as covered in section 5.1. Managing both the size and frequency of modification of active inequality constraint set in a way that achieves the best overall system performance is an area we are still investigating.

4 Geometrically Aware Object Manipulation System

In this section, we present an overview of our manipulation system. We describe object representations, the formulation of a nonlinear optimization problem representing manipulation, and user interaction schemes supported by our implementation.

4.1 Object Representation and Coordinates

Object representations fall into two general categories, reduced coordinates and unreduced, or full, coordinates. 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. A state vector q which combines all object's 7 state variables uniquely define the state of entire configuration. 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.

4.2 Interactive Manipulation as a Nonlinear Optimization Problem

Instead of searching for a solution of the inverse kinematics problem, we cast object manipulation as a constrained nonlinear optimization problem. Considering the fact that there is an infinite number of solutions for the under-determined case and we want to get a reasonably close solution when an exact solution is not available, nonlinear optimization has a number of advantages. De-

pending on the situation, we can experiment with different objective functions while still maintaining kinematic constraints. And, incorporation of inequality constraints turns out not to be very difficult in our optimization approach. Equation 2 represents an objective function to achieve the "Principle of Least Astonishment"[6] which suggests the system should move as little as possible from the previous configuration.

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

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:

$$\begin{array}{ll} minimize & E(q) \qquad q = q_1, q_2, .., q_n \\ s.t. & \Phi_i(q) = 0, \quad i = 1..m_{eq}, j = 1..m_{ineq} \\ & \Phi_j(q) > 0, \qquad m_{eq} + m_{ineq} = m \end{array}$$
$$(3)$$

Equality and Inequality Constraints

The inequality constraints used to achieve geometry awareness were described in Section 3. Goal and kinematic constraints are usually equality, though we employ inequality goal constraints in some cases. Sample constraints include:

- spherical joint : $r_1 + A_1 c_1 - r_2 - A_2 c_2 = 0$, 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: $n_2 \cdot ((r_1 + A_1 c_1) - p_2) = 0$, where n_2 is the surface normal and p_2 a point on the surface.

- point goal: $r_1 + A_1 c_1 - p_{goal} = 0$, where p_{goal} is the goal position for the specified object 1 point.

4.3 Interactive Object Manipulation Interface

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
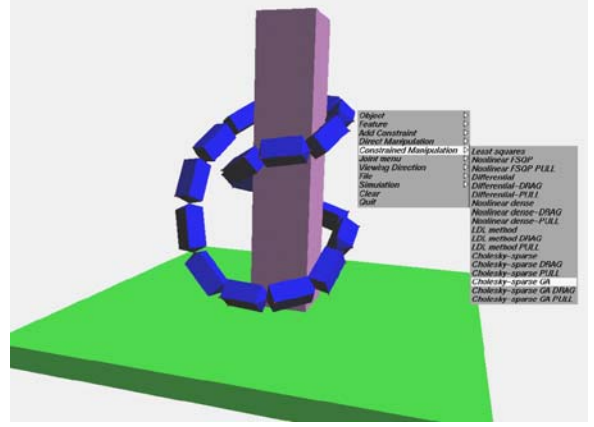

Figure 6: Snapshot from interactive object manipulation interface

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. 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.

Figure 6 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.

5 Solution Method for 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 re-

placed by equalities as described in Section 5.3, the first order necessary condition for the existence of a local minimizer $q*$ of the objective function of Equation 2 requires the existence of a Lagrange multiplier λ^* satisfying Karush-Kuhn-Tucker(KKT) condition

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

Combining this with the original constraints yields

$$F \begin{pmatrix} q \\ \lambda \end{pmatrix} = \left\{ \begin{array}{c} q - q_o - J(q)^T \cdot \lambda = 0 \\ \Phi_i(q) = 0 \end{array} \right\} \quad (5)$$

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. To solve the nonlinear system, we use Newton's method which converges quadratically if the starting point is within a certain range[8].

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

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

Since we have a good starting point, usually less than 3 iterations of system solving suffices the termination condition that requires 2 norm of all constraint errors should be less than 1.0E-5.

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

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

and multiply the upper row by J, then

$$\begin{array}{c} J\delta_1 - JJ^T\delta_2 = Jb_1 \\ -J\delta_1 = b_2 \end{array} \quad (9)$$

leading to

$$JJ^T\delta_2 = -b_2 - Jb_1 \quad (10)$$

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^T can 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^T even when J is sparse, due to the existence of single dense column.

We showed how to alleviate the problem of dense columns in the normal equation approach by employing the modified Schur complement method[7]. When dense columns occur, we rearrange and expand the system. When there are no dense columns, 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 [15] if the upper left corner block of the linear system 8 is a diagonal matrix.

5.1 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. The problem of choosing an ordering that minimizes fill-in is NP-hard[2]. Our system uses the approximate minimum degree heuristic (AMD) proposed by Davis, Amestoy, and Duff[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.

5.2 The Push Cholesky Algorithm

After the symbolic phase that concentrates about the pivoting rule and pivoting order, the numerical phase solves a sparse symmetric system. Our sparse factorization method is based on the push Cholesky algorithm[2]. $L_{.k}$ denotes the kth column of L and $L_{(i:j)k}$ denotes the kth column from row i to row 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

The push Cholesky algorithm is categorized as a column-Cholesky factorization in which the columns of L are computed in turn, one by one. 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 book-

keeping 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 having the same non-zero pattern below the diagonal of the previous column. By regrouping columns using the supernode technique, all columns in the supernode update exactly the same position in the subsequent columns. The supernode also makes loop unrolling possible [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.

5.3 Inequality Constraints

A common way to handle inequality constraints in nonlinear optimization is to add a logarithmic barrier function. As in 11, we take the logarithm of all inequality constraints and subtract the resulting terms from the original objective function.

$$
\begin{aligned}
minimize\ \|q - q_0\|^2 - \mu \sum \log \Phi_j(q) \\
s.t.\ \Phi_i(q) = 0 \\
i = 1..m_{eq},\ j = 1..m_{ineq},\ m_{eq} + m_{ineq} = m
\end{aligned}
\tag{11}
$$

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

$$
\begin{bmatrix} W & -J^T \\ -J & 0 \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}
$$
$$
W = \nabla^2 E(q) + \mu \sum \left[\frac{1}{\Phi_j^2(q)} \nabla \Phi_j(q) \nabla \Phi_j(q)^T \right. \tag{12}
$$
$$
\left. - \frac{1}{\Phi_j(q)} \nabla^2 \Phi_j(q) \right]
$$

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 13 shows how to eliminate problems related to the barrier function approach.

$$
\begin{aligned}
minimize & \quad E(q, Y) \\
s.t. & \quad \Phi_i(q) = 0 \\
& \quad \Phi_j(q) - Y_j^2 = 0
\end{aligned}
\tag{13}
$$

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.

5.4 Over-determined case

Conflicting or unreachable goals yield 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. The main disadvantage of this approach is that the overall structure of the system has to be modified to accommodate new objective function. 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 how to formulate and manage a set of inequality constraints that enables 3D object manipulation with geometry awareness. Based on the nearest feature witness and interpenetration information reported by a geometry system, inequality constraints for preventing interpenetration are formulated. The geometric constraint monitoring system updates inequality constraints as objects moves and maintain active constraints set as small as possible. Combined with kinematic and goals constraints, the object manipulation problem is cast as a nonlinear optimization problem. We developed an efficient sparse algorithm for solving the optimization

problem. Based on these results, we implemented a program that allows users to manipulate complex object configurations interactively. The performance of the manipulation algorithm scales well with the number of objects and active equality and inequality constraints; the push Cholesky algorithm performs well and scales nearly linearly, for instance. 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. System performance does not degrade in the presence of closed loop configurations or inequality constraints.

Some of our current work aims at developing more powerful and appropriate interaction metaphors for manipulating complex 3D objects. At present, our system employs the mouse to define and manipulate simple point-based goals and trajectories. The human figure manipulation "challenge problem" of Figure 1 involves contact over extended regions and is tedious to carry out using simple manipulation metaphors. Furthermore, it cannot even be solved fully adequately without accounting for object deformability. Extending geometrically aware object manipulation to include deformability is thus an important remaining challenge.

7   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 T. Terlaky, editor, *Interior point methods of mathematical programming*, pages 189–252. Kluwer, Dordrecht, The Netherlands, 1996.

[4] 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.

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

[6] 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.

[7] Min-Hyung Choi and James F. Cremer. Interactive manipulation of articulated objects with geometry awareness. In *Proceedings of IEEE International Conference on Robotics and Automation*, May 1999.

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

[9] M. Gleicher. *Differential Approach to Graphical Manipulation*. Ph. D. thesis, Carnegie Mellon University, 1994.

[10] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. *Computer Graphics (SIGGRAPH 96)*, 30(2):171–180, 1996.

[11] Brian Mirtich. Rigid body contact: Collision detection to force computation. Technical Report TR-98-01, Mitsubishi Electrical Research Laboratory, 1998.

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

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

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

[15] 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.

[16] Xinmin Zhao and Norman Badler. Near real-time body awareness. In *Computer Aided Design*, 1994.