Using Degrees of Freedom Analysis To Solve Geometric Constraint Systems

Glenn A. Kramer[†]
Schlumberger Laboratory for Computer Science
8311 North RR 620
Austin, Texas 78720-0015
email: gak@slcs.slb.com

Abstract

This paper address the problem of finding the positions and orientations of a set of rigid bodies that satisfy a set of geometric constraints. Solving geometric constraint systems occurs in such tasks as simulating the behavior of a collection of mechanical parts, shape optimization, tolerance analysis, and assembly planning. Such problems are traditionally solved by reformulating the geometry and constraints as algebraic equations which are then solved symbolically or numerically. But many such problems can be solved by reasoning symbolically about the geometric bodies themselves using a new technique called degrees of freedom analysis. In this approach, a sequence of actions is devised to satisfy each constraint incrementally, resulting in a monotonic decrease of the system's available degrees of freedom. This sequence of actions solves, in a maximally decoupled form, the equations resulting from an algebraic representation of the problem. Degrees of freedom analysis has significant computational advantages over conventional algebraic approaches. While symbolic algebraic solution of the constraints takes exponential time, degrees of freedom analysis takes polynomial time. The numerical solution of the equations using explicit geometric reasoning is significantly faster than by using standard iterative techniques. The utility of the technique is demonstrated with a program that assembles and kinematically simulates mechanical linkages.

Introduction

Solving geometric constraint systems is an important problem with applications in many domains, for example: describing mechanical assemblies, constraint-based sketching and design, geometric modeling for CAD, and kinematic analysis of robots and other mechanisms. An important class of such problems involves finding the positions, orientations, and dimensions of a set of geometric entities that satisfy a set of geometric constraints. This paper first examines traditional means of solving geometric constraint satisfaction problems (GCSP's). Then, it introduces a fundamentally different philosophy of constraint satisfaction, based on symbolic geometric reasoning and an operational semantics for constraint satisfaction.

The notion of providing an operational semantics for geometric constraint satisfaction is relatively new, and is not widely used. Previous attempts at solving large constraint systems using this notion have either relied on the user of the system to specify the sequence of operations [Rossignac, 1986], or have used 'weak' methods such as message passing [Wang, in preparation]. The methods described in this paper are almost anthropomorphic, and can be computed automatically with no human intervention.

The approach described in this paper is applicable to a broad class of geometric constraints. The approach is illustrated in the context of the kinematic simulation of mechanical linkages. The geometric constraints encountered in kinematics are applicable to other tasks such as tolerance analysis, assembly planning, and constraint-based design.

Kinematic analysis presents interesting challenges due to the intimate role that complex 3D geometry plays in the behavior of mechanisms. While algebraic methods are dominant in mechanism analysis, purely geometric methods are also used because they 'maintain touch with physical reality to a much greater degree than do the algebraic methods' and 'serve as useful guides in directing the course of equations' [Hartenberg and Denavit, 1964]. Capturing this intuition and putting it in algorithmic form leads to increased efficiency in the manipulation of the algebraic equations.

This paper describes how to use geometric reasoning to guide the solution of the sets of complicated nonlin-

[†]Part of this research was conducted at the School of Cognitive and Computing Sciences at the University of Sussex, Brighton, East Sussex, England BN1 9QH.

ear equations that arise in mechanism simulation. A program called TLA embodies this methodology. It simulates a mechanism by first reasoning at the geometric level about how to assemble it. TLA then uses this assembly plan as a metaphor to solve the equations in a stylized, highly decoupled manner. Efficient solution is important because these equations are solved repeatedly in tasks such as simulation and optimization. The approach described in this paper greatly reduces the computational complexity of solving such systems.

Terminology

The objects of interest in solving GCSP's are called geometric *entities*, or geometric *objects*; some examples are lines, circles, and rigid bodies. Entities have degrees of freedom, which allow them to vary in location or size. For example, in 3D space, a general rigid body has three translational and three rotational degrees of freedom. A circle with a variable radius has three translational, two rotational, and one dimensional degree of freedom (a third rotational degree of freedom is not required because the circle is invariant under the rotation about its axis).

The *configuration variables* of a geometric object are defined as the minimal number of real-valued parameters required to completely specify the object in space. The configuration variables are used to parameterize an object's translational, rotational, and dimensional degrees of freedom (DOF's), with one variable required for each DOF. A *configuration* of an object is a particular assignment of the configuration variables, yielding a unique instantiation of the geometric entity.

The definition of a GCSP is then as follows: Given a set of geometric entities and constraints between them, find the values of the configuration variables of the objects such that all constraints are satisfied. The collection of entities and constraints is called the *constraint system*, or simply the *system*.

Kinematic constraints

The initial application domain of mechanical linkages was chosen because the constraints in that domain are applicable to a wide variety of other tasks. The application domain is restricted to rigid bodies; therefore, bodies have translational and rotational DOF's, but no dimensional DOF's.

Constraints describing joint behavior in mechanical devices can be modeled as relationships between sets of distinguished points on different bodies. A *marker* consists of a point in 3D space, along with two orthogonal axes, z and x, which emanate from the point. The position of a marker is the position of its point, while its orientation is determined by its axes. Since all bodies are rigid, constraints between markers constrain the bodies to which they are attached. The constraints between pairs of markers m_1 and m_2 are:

coincident(m_1, m_2): Markers m_1 and m_2 are spatially coincident.

in-line(m_1, m_2): m_1 lies on the line through m_2 parallel to m_2's z axis.

in-plane(m_1, m_2): m_1 lies in the plane through m_2 normal to m_2's z axis.

parallel-z(m_1, m_2): The z axes of markers m_1 and m_2 are parallel.

offset-z(m_1, m_2, α): The z axes of m_1 and m_2 make an angle of $\alpha \neq 0$.[1]

offset-x(m_1, m_2, α): The z axes of m_1 and m_2 are parallel; and the angle from m_1's x axis to m_2's x axis is α.

helical(m_1, m_2, δ): The z axes of m_1 and m_2 are parallel; and the angle from m_1's x axis to m_2's x axis is linearly related to the distance between m_1 and m_2 by a pitch constant δ.

Combinations of these constraints, relating markers on different rigid bodies, may be used to model all of the 'lower pair' joints described by Reuleaux [Reuleaux, 1876]. For example, a revolute joint, which allows one rotational degree of freedom between two bodies, is modeled with a **coincident** constraint and a **parallel-z** constraint. A translational, or prismatic, joint is modeled with an **in-line** constraint and a **offset-x** constraint. Some types of higher pairs may also be modeled with the above constraints, for example, the 'universal' joint and 'slotted pin' joint. The constraints defined above are sufficient to describe all mechanical linkages as well as many static mechanical assemblies.

Equational techniques for solving GCSP's

GCSP's are usually solved by modeling the geometry and constraints with algebraic equations that relate the configuration variables of the different objects according to the problem constraints. Solving these equations—either numerically or symbolically—yields the desired configuration for each geometric entity.

Numerical solution

Numerical solutions represent constraints using *error terms*, which vanish when the constraint is satisfied, and otherwise have magnitude proportional to the degree to which the constraint is violated. The *error function* is the sum of all error terms; the constraint system is satisfied when the error function is zero. One of the most efficient methods for finding a zero of the error function is Newton-Raphson iteration [Press *et al.*, 1986].

Numerical techniques find zeros of the error function by 'sliding' down the function's gradient. This process

[1] This constraint restricts one rotational DOF (*i.e.*, it leaves the system of two markers with two rotational DOF's). If α were to be zero, the system of two markers would have only a single rotational DOF, describing the qualitatively different state obtained by using the **parallel-z** constraint.

is necessarily iterative for nonlinear problems. Numerical techniques have many drawbacks. Each iteration of Newton-Raphson is slow, taking $O(c^3)$ time, where c is the number of constraints. In addition, the Jacobian matrix must be evaluated at every iteration. Overconstrained situations, which are quite common, require pre- and post-analysis to remove redundant constraints before solving and to check them later for consistency. Newton-Raphson can jump chaotically between different roots of the error function during solution [Peitgen and Richter, 1986], which can make the choice of initial solution guess crucially important. Underconstrained situations require pseudo-inverse techniques, since the constraint matrix is non-square. Additionally, when a solution is impossible, no information is available to pinpoint the smallest set of constraints which are inconsistent.

Symbolic solution

Symbolic solutions use algebraic rewrite rules or other techniques to isolate the configuration variables in the equations in a predominantly serial fashion [Buchberger *et al.*, 1983]. Once a solution is found, it may be reused—or *executed*—on topologically equivalent problems. Execution is fast, typically linear in the number of constraints. If numerical stability is properly addressed, the solution can be more accurate by virtue of being analytic; there is no convergence tolerance as found in numerical techniques. The principal disadvantage of symbolic techniques is the excessive (potentially exponential) time required to find a solution or determine that one does not exist [Liu and Popplestone, 1990]. Poorly-chosen configuration variable assignments can exacerbate the problem by coupling the equations in unnecessarily complicated ways, requiring very clever and complex inferences. Hence, the symbolic techniques are feasible and complete only for very small problems.

Geometric techniques for solving GCSP's

The geometric approach to solving GCSP's relies on a representation shift from reasoning about configuration variables to reasoning about the DOF's of the actual geometric entities. Configuration variables are related to each other by sets of equations that may be very complicated, tightly coupled, and highly nonlinear; in addition, the domains of the configuration variables are continuous, yielding an infinite search space. In contrast, the degrees of freedom of an object form a compact, discrete-valued, linear description of the state of the object. Coupling of degrees of freedom is rarely encountered, and when it does occur, it can be accommodated easily.

Degrees of freedom form abstract equivalence classes describing the state of a geometric entity without specifying how the constraints that lead to that state are satisfied. DOF's are grouped into three equivalence classes: rotational, translational, and dimensional. All DOF's of the same type are considered identical elements of

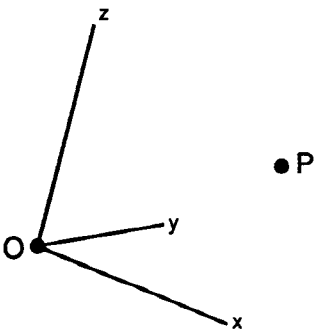

Figure 1: A rigid body with two embedded points.

that resource. DOF resources are consumed by moving an object so as to satisfy a constraint. Further actions are then confined to those that do not violate any previously-satisfied constraints. Therefore, every constraint, upon being satisfied, introduces invariant quantities for the satisfaction of subsequent constraints, and reduces the number of remaining degrees of freedom.

Measurements and actions form the basis for an operational semantics for constraint satisfaction. For example, consider points A and B on a line L, where L has no constraints applied to it. Suppose a constraint specifies that point A be coincident with a fixed point C. The line may be translated to make those two points coincident. If another constraint, say one involving point B on the line, is solved next, any action applied to line L must preserve the location of point A. Therefore, subsequent actions are limited to rotations and scaling about point A. The constraint **coincident**(A, C) removes the line's translational DOF's, thereby restricting subsequent operations. A similar operational semantics is found in [Wang, in preparation].

Reasoning about degrees of freedom is essential to decoupling constraints. Consider the xyz coordinate frame in Figure 1, with points O, at the origin, and P, in some arbitrary location, rigidly fixed in the coordinate frame. The coordinate frame is parameterized by six configuration variables, three for the translational DOF's, and three for the rotational DOF's. Thus, the coordinate frame is free to translate and rotate in space.

Fixing the position of either point O or P (through the satisfaction of some constraint) removes the three translational DOF's in the system: the coordinate frame may only rotate about the fixed point in order to satisfy subsequent constraints. But consider the constraints in terms of configuration variables. Fixing the position of point O uniquely determines the three translational configuration variables, while fixing the position of P introduces nonlinear constraint equations into the system to relate the configuration variables to the distance $\overline{OP}$. Solving constraint systems in the configuration variable space is difficult because of this type of coupling between configuration variables. Solving in DOF space is simpler because the actions can be specified indepen-

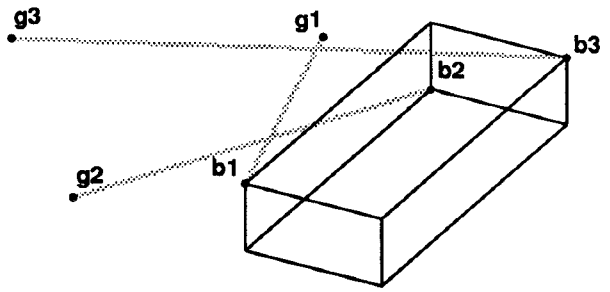

Figure 2: A brick with three **coincident** constraints.

dently of how the system is parameterized in terms of configuration variables.

The use of the metaphors of measurement and action to guide equation solution distinguishes the approach described here from other techniques for solving large sets of nonlinear equations. Since the DOF representation is decoupled, a monotonic decrease in the degrees of freedom in a system can be achieved as the constraints are incrementally satisfied, leading to polynomial-time algorithms for constraint satisfaction.[2]

Degrees of freedom analysis

This section introduces the philosophy of degrees of freedom analysis through two examples. The TLA program's solution of each example is illustrated. The first example involves constraints whose solution can be linearized, while the second example involves interacting constraints that must be solved simultaneously.

Example 1: the brick

In this problem, the brick of Figure 2 must be configured to satisfy the three **coincident** constraints graphically depicted as the grey lines between the brick's markers $b1, b2, b3$ and the desired locations, denoted by markers $g1, g2, g3$ fixed in the global coordinate frame. Equations can be developed to relate the configuration variables of the brick's coordinate frame to those of the global coordinate frame. The numerical solution of these equations, using Newton-Raphson, is illustrated graphically in Figure 3. Each of the grey outlines represents an intermediate configuration of the brick during the solution. The figure depicts only every third iteration of Newton-Raphson, and step sizes were clipped to improve the behavior of the algorithm.

[2]It should be noted that the plan of measurements and actions that satisfy the constraint network do not necessarily correspond to a physically-realizable plan for assembling a collection of real objects. Since the objects in a GCSP are purely geometric, they have no volume or other physical properties. Objects may pass through each other in a ghost-like fashion, on their way to satisfying constraints. This property of the solution process allows decoupling the solution of all constraints affecting any one entity.

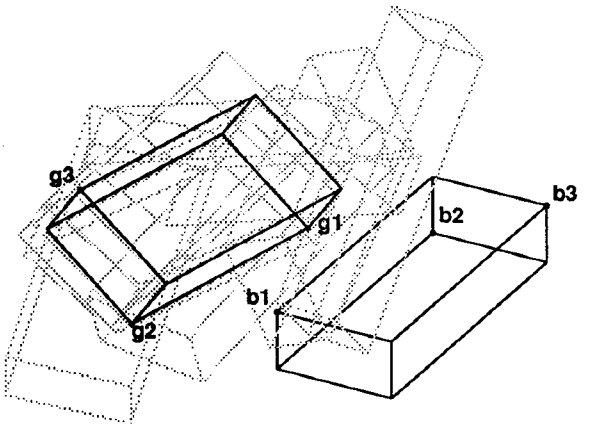

Figure 3: Brick solution using Newton-Raphson.

The TLA program solves the brick problem more efficiently by using geometric knowledge to satisfy the constraints incrementally. The solution is shown in Figure 4. TLA assumes that initially the brick is free to move anywhere; it just happens to be in the given initial configuration **C0**. To satisfy **coincident**$(b1, g1)$, TLA translates the brick by the vector from $b1$ to $g1$, leaving the brick in configuration **C1**. To ensure **coincident**$(b1, g1)$ remains satisfied, all further actions that move the brick must be rotations about $g1$, *i.e.*, the brick has only its rotational degrees of freedom left. To satisfy **coincident**$(b3, g3)$, TLA measures the vector $v1$ from $g1$ to $b3'$ (where $b3$ has been moved by the previous translation) and vector $v2$ from $g1$ to $g3$. These two vectors are shown as dashed lines in Figure 4. Then TLA rotates the brick about $g1$ around vector $v1 \times v2$ by the angle between $v1$ and $v2$, to configuration **C2**. This satisfies **coincident**$(b3, g3)$ without violating **coincident**$(b1, g1)$. This action also removes two of the remaining rotational degrees of freedom; in order to preserve the two already-satisfied constraints, all future actions must be rotations about $v2$. To satisfy the final constraint, TLA drops perpendiculars from $b2''$ to $v2$, and from $g2$ to $v2$, and rotates the brick about $v2$ by the angle between the perpendiculars. This brings the brick to its final configuration. The solution is very deliberate, as opposed to the meandering of the numerical approach of Figure 3. The sequence of actions performed above constitute a plan for moving the brick from an arbitrary position to one satisfying the constraints.

For this part of the problem solution, TLA reasons only about geometry, actions and degrees of freedom. No equations are developed, and no model requiring configuration variables or other abstract state is needed. Constraints are satisfied by measuring the brick's geometric properties (often using additional geometric constructions) and then moving it. The brick-moving plan derived using this method is next used to solve for the brick's configuration variables as represented in a computer; this may be done regardless of how the local coordinate frame of the brick is described. All that is re-

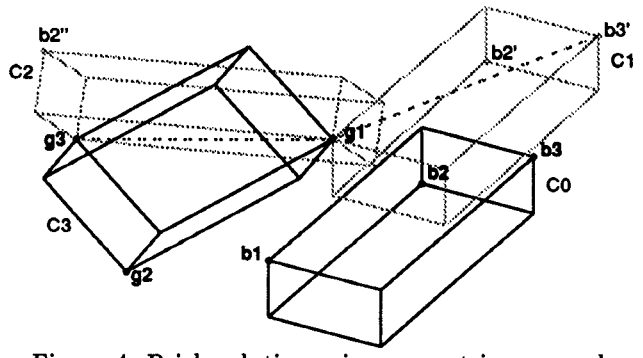

Figure 4: Brick solution using geometric approach.

quired is a set of operators for translating and rotating rigid bodies, and a set of functions that can measure, relative to a global coordinate system, points and vectors attached to any rigid body. These capabilities are provided by homogeneous coordinate transforms, which most 3D graphics and robotics systems use [Snyder, 1985], or dual quaternions [Hamilton, 1969].

The plan, when executed, becomes a metaphor for solving the equational representation of the constraint system.[3] By using the primitive actions of translation and rotation, the plan effectively decouples the equations into small independent sets that can be solved analytically.[4] As new constraints are satisfied, previously satisfied constraints (which may correspond to complicated relations between configuration variables) become invariants for later steps in the solution. Geometry, as used in the plan, provides the vocabulary and operators that allow preserving these invariants. The use of the assembly plan to guide equation solution distinguishes TLA from other programs that solve large sets of nonlinear equations.

Maintaining knowledge about DOF's

TLA keeps track of the number and types of degrees of freedom each body has as it solves a problem. It represents this information with predicates of the following form:

body-has-*n*-TDOF(*body, arg1, arg2, ...*)
body-has-*n*-RDOF(*body, arg1, arg2, ...*)
where $n \in \{0, 1, 2, 3\}$

In these predicates, TDOF stands for translational degrees of freedom, and RDOF for rotational degrees of freedom. The arguments *arg1, arg2, ...* specify any fixed points or axes on the bodies that restrict their freedom. Initially, every body in the system except the grounded body has 3 TDOF and 3 RDOF. As actions are taken to satisfy constraints, the bodies in the system lose some of their degrees of freedom. When all bodies have 0 TDOF and 0 RDOF, the problem is solved.

At each step in solving for a body's configuration, TLA must know what action to take given the body's current constraints, and how that action further reduces the body's degrees of freedom. This information is stored in a *plan fragment table*. Conceptually, the plan fragment table is a three-dimensional dispatch table, indexed by TDOF, RDOF, and constraint type. Each entry in the table specifies how to move the rigid body to satisfy the new constraint using only available degrees of freedom, and what degrees of freedom the body will have after the action is performed. The plan fragment table contains information about how to satisfy constraints when one of the markers participating in the constraint has its appropriate attributes fixed, or *globally known*. Thus, a globally known position of one marker is required for solving a **coincident** constraint, and a globally known z axis is needed to solve a **offset-z** constraint.

In the brick example, the first constraint to be satisfied is arbitrarily chosen to be **coincident**($b1, g1$). The global position of $g1$ is known. Initially the brick has 3 TDOF and 3 RDOF; thus the index into the plan fragment table is (3, 3, **coincident**). This entry contains the following information (modified for readability):

Initial state:
body-has-3-TDOF(*body*)
body-has-3-RDOF(*body*)

Plan fragment:
begin
translate(*body*, **vector-difference**(**gmp**(*M1*), **gmp**(*M2*)));
end;

New state:
body-has-0-TDOF(*body*, **gmp**(*M2*))
body-has-3-RDOF(*body*)

Explanation:
Body *body* is free to translate. A **coincident** constraint must be satisfied between marker *M1*, whose global position is known, and marker *M2* on *body*. Therefore *body* is translated by the vector from the current global position of *M2* to the known global position of *M1*. This action removes all three translational degrees of freedom.

[3] One way of viewing the computation is for the geometric bodies to be actually moving in the computer's virtual world as the constraints are satisfied. However, a more "pure" view of the computation is as solving a set of equations, independent of the interpretation of the solution steps as geometric transformations. Since the goal of solving GCSP's is to find the configuration variables of the system, and not to plan any actions to move objects through space, the notion of action is being used merely to derive an efficient solution to the constraint equations; in this sense, the plan is a metaphor for equation solution.

[4] Not all problems may be solved analytically; some require iterative solutions. In such cases TLA fails in the plan construction phase. It is possible, however, to use the information from the failure to reduce significantly the dimensionality of the iterative problem that must be solved. See [Kramer, 1990] for details.

The variable *body* is bound to the object representing the brick. The initial state of the body is that it has all six of its degrees of freedom; it is free to translate and rotate through space. The variable $M1$ gets bound to the globally known marker (*i.e.*, $g1$), while variable $M2$ is bound to the underconstrained marker in the **coincident** constraint being satisfied (*i.e.*, $b1$). The plan fragment specifies how to move the body to satisfy the constraint (the function name **gmp** stands for "global marker position"). In the specification of the new state, the predicate **body-has-0-TDOF** has an additional argument which specifies the point on the body which is constrained to be stationary. The textual explanation — with variable names replaced by their bindings — helps the user to understand the solution process.

The next constraint satisfied in the brick example is **coincident**$(b3, g3)$. Since the brick now has 0 TDOF and 3 RDOF, the index into the plan fragment table is $\langle 0, 3, \textbf{coincident}\rangle$. The plan fragment in that entry specifies how to rotate a body with 0 TDOF, 3 RDOF to satisfy a **coincident** constraint, and specifies that the new state of the body is 0 TDOF, 1 RDOF. The process continues until all constraints are satisfied.

For the kinematic constraints defined in this paper, there are approximately one hundred entries in the plan fragment table; some plan fragments are quite simple, like the one described above, while others involve more complex calculations and conditionals to handle potential mathematical degeneracies. The complete plan fragment table appears in [Kramer, 1990].

Example 2: interacting bodies

Bodies rarely interact exclusively with fixed points, as in the brick example. Often, they interact with other partially constrained bodies. In Figure 5 body **A** is constrained to 0 TDOF, 1 RDOF by the constraints **coincident**$(a1, g1)$ and **parallel-z**$(a1, g1)$. TLA infers that marker $a2$ must lie on a circle about $a1$. Body **B** is similarly constrained. To satisfy **coincident**$(a2, b2)$, TLA intersects the circles to find the two globally acceptable locations for the markers. TLA distinguishes the locations with a branch variable **q**. A user of TLA chooses which solution to use by specifying the value of **q**. TLA places a 'pseudo-marker' p at this location; this is a marker which is not part of the original problem specification, but is introduced during the problem solution.

With the intersection point defined, TLA satisfies the **coincident** constraint for bodies **A** and **B** independently. It does this by introducing the constraints **coincident**$(p, a2)$ and **coincident**$(p, b2)$. Since p's position is globally known, the plan fragment table may be used to find the appropriate actions to satisfy the two introduced constraints. When they are satisfied, **coincident**$(a2, b2)$ is also satisfied.

In this manner, local information, in the form of loci of points on partially constrained bodies, may be combined through locus intersection to yield information about globally permissible locations of points. Pseudo-markers denote these intersections, and auxiliary constraints are introduced to relate the partially constrained markers to the pseudo-marker. Then the plan fragment table is used to find the appropriate actions to satisfy the constraints.

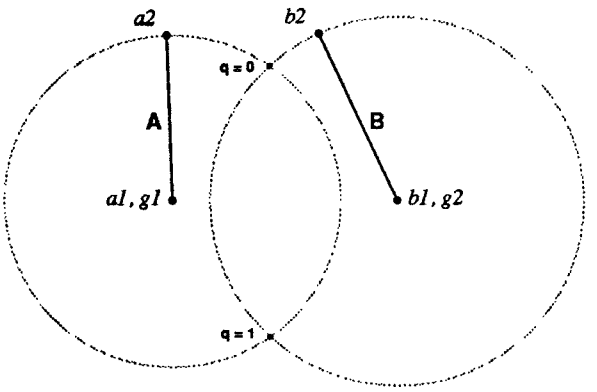

Figure 5: Solving for two interacting bodies (z axes point out of the page).

Maintaining knowledge of loci

TLA uses a *locus table* to specify the loci to which partially constrained markers are confined. Loci are determined completely by the degrees of freedom that a body has. For example, all markers on a body with 0 TDOF, 1 RDOF are constrained to lie on circles around the body's fixed point. Markers on a body with 0 TDOF, 3 RDOF must lie on spheres, and markers on a body with 2 TDOF, 0 RDOF must lie in planes.

A *locus intersection table* allows TLA to know when enough information is known about sets of partially constrained markers to determine their configurations fully. This table has entries for all pairs of shapes in the locus table. For example, a sphere intersected with a circle yields at most two discrete points (except in the degenerate case of the circle lying on the sphere); a plane intersected with a cylinder yields an ellipse. For the constraints described in this paper, all loci are analytically describable, as are all pairwise intersections of loci.

Theoretical and empirical analysis

Space does not permit a full description of the algorithms used in degrees of freedom analysis; details may be found in [Kramer, 1990]. The high-level outline of the algorithm is described below. Give a constraint graph delineating the rigid bodies and the constraints that pertain to them:

1. Identify rigid substructures (chains or loops) in the constraint graph.
2. For a given rigid substructure, construct a plan (a sequence of measurements and actions) to assemble the substructure.

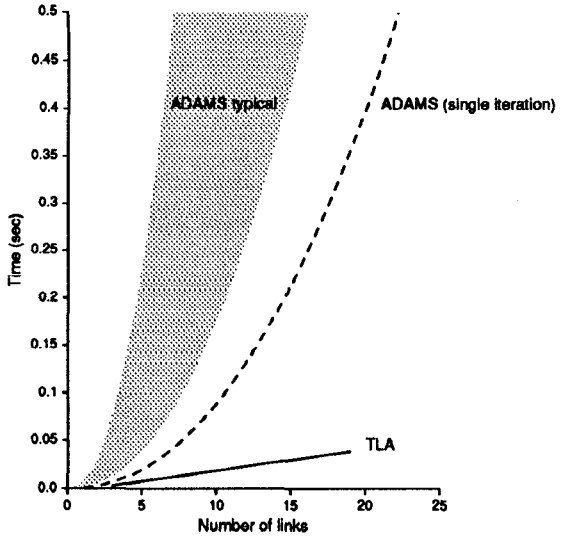

Figure 6: Timing comparisons of TLA and ADAMS.

3. Rewrite the chain or loop as a single node in the constraint graph.

4. Repeat the above steps until the entire graph has been reduced to a single node.

Using this algorithm, a plan to satisfy a GCSP is generated in $O(gc)$ time, where g is the number of geometric entities in the constraint system, and c is the number of constraints. The plans may be executed in $O(g \log g)$ time, although typically the execution time is linear in g.

An important aspect of the algorithm is that it is *canonical*. Whenever there is a choice among actions to take in satisfying constraints, any action may be chosen with the confidence that no backtracking will be required later. This property is essential to ensuring the polynomial complexity of the algorithm.

The TLA program has also been empirically compared with the ADAMS mechanism simulator, which employs iterative numerical solution techniques [ADAMS, 1987]. The graph of Figure 6 shows the runtimes of ADAMS and TLA as a function of the number of bodies in a mechanism. The dashed line shows the time per iteration for ADAMS; typically, between 2 and 12 iterations are required to solve a GCSP, as indicated by the gray area. In contrast, the behavior of TLA is linear, and is substantially more efficient.

TLA has simulated dozens of complex planar and spatial mechanisms; the largest example is a sofa-bed, shown in Figure 7. This mechanism has 16 rigid bodies (or links), 22 joints, and two driving inputs, and is described by 115 algebraic constraints, 19 of which are redundant. The assembly plan generated by TLA is stored as a program which is 655 lines of Lisp code. This Lisp program runs almost two orders of magnitude

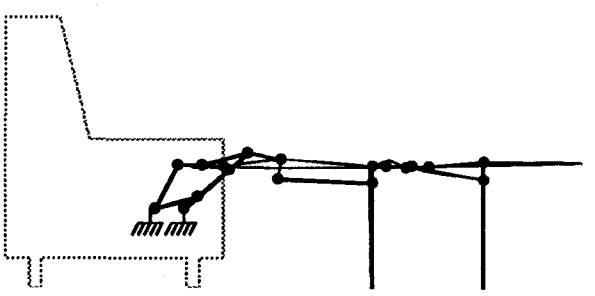

Figure 7: Sofa-bed mechanism (extended).

faster than iterative numerical techniques embodied in ADAMS, when used to solve the same set of constraints.[5]

Discussion

Algebra has long been the *lingua franca* of science and engineering, but it can provide only a partial appreciation of the actual domain under study. An understanding of geometry is essential to solving problems insightfully and efficiently in the mechanical world. TLA demonstrates this for the task of mechanical assembly and simulation. By using geometry to guide equation solving, TLA provides orders of magnitude speedup over 'general-purpose' mathematical techniques. This means that interactive tools for the simulation, optimization, and synthesis of complex mechanical devices become feasible [Kramer and Barrow, 1989].

TLA currently reasons only about rigid bodies. We are in the process of expanding this research to include more general geometric objects with dimensional constraints as well. In addition, we plan to explore tolerancing and dynamics issues in a geometric context.

Related work

Using degrees of freedom analysis to generate an assembly plan, and using the resulting plan as a metaphor to guide equation solution both appear to be novel features of TLA. The use of actions to satisfy constraints is found in [Wang, in preparation] and [Rossignac, 1986], but the solution methods are somewhat weaker.

Sketchpad [Sutherland, 1963] and ThingLab [Borning, 1979] represented geometric constraints equationally, relying on relaxation for nonlinear equations. Popplestone *et al.* explored, with limited success, solving assembly problems algebraically using some geometric guidance [Popplestone *et al.*, 1980]. More recently Popplestone has focused on using group theory to represent geometric symmetries [Popplestone, 1987]. This work could profitably be incorporated into TLA. Faltings [Faltings, 1989] and Joskowicz [Joskowicz, 1987] are investigating deriving kinematic constraints directly from

[5] In fact, while the time per iteration could be measured for ADAMS, numerical stability problems kept ADAMS from converging in this particular example.

geometry. Such a facility would free the user of TLA from having to model a mechanism in terms of abstract concepts like markers.

Summary

It seems unlikely that there will ever be an efficient solution technique to general constraint problems. However, this research may be viewed as a step toward efficient solution of a specific type of constraint satisfaction problem. While it provides leverage in solving GCSP's, it is doubtful that the method will extend very far beyond the realm of geometry.

There are some broad concepts that can be reused in formulating constraint satisfaction problems for other domains. The notion of abstracting some continuous space (*e.g.,* position and orientation) into a discrete space (*e.g.,* degrees of freedom) may apply to other domains. Designing algorithms that exploit monotonic trends (such as the reduction of degrees of freedom of a geometric entity) can lead to polynomial-time algorithms. Creative representation shifts will be required to use these principles in other domains, but if they can be found, the benefits may be substantial.

Acknowledgments

Phil Agre helped implement the latest version of TLA, and contributed technically in many ways. I would also like to thank Harry Barrow, David Barstow, David Gossard, Walid Keirouz, Reid Smith, Bob Young, and David Ullman.

References

[ADAMS, 1987] Mechanism Dynamics, Inc., Ann Arbor, Michigan. ADAMS *Users Manual*, 1987.

[Borning, 1979] Alan H. Borning. *Thinglab: A Constraint-Oriented Simulation Laboratory.* PhD thesis, Stanford University, Stanford, California, July 1979.

[Buchberger *et al.*, 1983] B. Buchberger, G. E. Collins, and R. Loos, editors. *Computer Algebra: Symbolic and Algebraic Computation.* Springer-Verlag, Wien, second edition, 1983.

[Faltings, 1989] Boi Faltings. Reasoning about kinematic topology. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Detroit, Michigan, August 1989.

[Hamilton, 1969] W. R. Hamilton. *Elements of quaternions.* Chelsea, New York, 1969.

[Hartenberg and Denavit, 1964] R. S. Hartenberg and J. Denavit. *Kinematic Synthesis of Linkages.* McGraw Hill, New York, 1964.

[Joskowicz, 1987] Leo Joskowicz. Shape and function in mechanical devices. In *Proceedings of the National Conference on Artificial Intelligence*, Seattle, Washington, August 1987.

[Kramer and Barrow, 1989] Glenn A. Kramer and Harry G. Barrow. New approaches to linkage synthesis. In *International Joint Conference on Artificial Intelligence* (video track), Detroit, Michigan, August 1989.

[Kramer, 1990] Glenn A. Kramer. *Geometric Reasoning in the Kinematic Analysis of Mechanisms.* Dphil thesis, University of Sussex, Brighton, UK, October 1990.

[Liu and Popplestone, 1990] Yanxi Liu and Robin J. Popplestone. Symmetry constraint inference in assembly planning: Automatic assembly configuration specification. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1038–1044, Boston, Massachusetts, 1990.

[Peitgen and Richter, 1986] H.-O. Peitgen and P. H. Richter. *The Beauty of Fractals: Images of Complex Dynamical Systems.* Springer-Verlag, Berlin, 1986.

[Popplestone *et al.*, 1980] R. J. Popplestone, A. P. Ambler, and I. M. Bellos. An interpreter for a language for describing assemblies. *Artificial Intelligence*, 14(1):79–107, August 1980.

[Popplestone, 1987] R. J. Popplestone. The Edinburgh Designer System as a framework for robotics or, the design of behavior. COINS Technical Report 87-47, University of Massachusetts, Amherst, Massachusetts, May 1987.

[Press *et al.*, 1986] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, Cambridge, UK, 1986.

[Reuleaux, 1876] M. M. Reuleaux. *The Kinematics of Machinery.* Macmillan & Co., New York, 1876. Translated by Alex B. W. Kennedy.

[Rossignac, 1986] J. R. Rossignac. Constraints in constructive solid geometry. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, pages 93–110. ACM Press, 1986.

[Snyder, 1985] Wesley E. Snyder. *Industrial Robots: Computer Interfacing and Control.* Industrial Robot Series. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

[Sutherland, 1963] Ivan E. Sutherland. *Sketchpad: A Man-Machine Graphical Communication System.* PhD thesis, MIT, Cambridge, Massachusetts, 1963.

[Wang, in preparation] Bin Wang. *An Operational Approach for Constraint Satisfaction.* PhD thesis, University of Southern California, Los Angeles, California, (in preparation).