The Motion of Planar Compliantly-Connected Rigid Bodies in Contact With Applications to Automatic Fastening

Bruce R. Donald*
Dinesh K. Pai**

TR 92-1296
July 1992

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This paper describes research done in the Robotics and Vision Laboratory and the Computer Science Department at Cornell University. Support for our robotics research is provided in part by the National Science Foundation under grants No. IRI-8802390, IRI-9000532 and by a Presidential Young Investigator award to Bruce Donald, and in part by the Air Force Office of Sponsored Research, the Mathematical Sciences Institute, Intel Corporation and AT&T Bell Laboratories.
**Supported in part by ONR Grant N00014-88K-0591, ONR Grant N00014-89J-1946, NSF Grant DMC-86-17355 at Cornell University and by NSERC Operating Grant OGP0122128 at UBC.

To Appear in *The International Journal of Robotics Research*

The Motion of Planar Compliantly-Connected Rigid Bodies in Contact with Applications to Automatic Fastening

Bruce R. Donald*
Computer Science Department
Cornell University

Dinesh K. Pai†
Computer Science Department
University of British Columbia

July 22, 1992

Abstract

We consider the problem of planning and predicting the motion of a flexible object amidst obstacles in the plane. We model the flexible object as a rigid "root" body, attached to compliant members by torsional springs. The root's position may be controlled, but the compliant members move in response to forces from contact with the environment. Such a model encompasses several important and complicated mechanisms in mechanical design and automated assembly: snap-fasteners, latches, ratchet and pawl mechanisms, and escapements. The problem is to predict the motion of such a mechanism amidst fixed obstacles. For example, our algorithm could be used to determine whether a snap-fastener design can be assembled with a certain plan.

In this paper we analyze the physics of these flexible devices, and develop combinatorially precise algorithms for predicting their movement under a motion plan. Our algorithms determine when and where the motion will terminate, and also computes the time-history of contacts and mating forces. In addition to providing the first known exact algorithm that addresses flexibility in motion planning, we also note that our approach to compliance permits an exact algorithm for predicting motions under rotational compliance, which was not possible in earlier work.

We discuss the following issues: the relevance of our approach to engineering (which we illustrate through the examples we ran using our system), the computational methods employed, the algebraic techniques for predicting motions in contact with rotational compliance, and issues of robustness and stability of our geometric and algebraic algorithms. Our computational viewpoint lies in the interface between differential theories

*This paper describes research done in the Robotics and Vision Laboratory and the Computer Science Department at Cornell University. Support for our robotics research is provided in part by the National Science Foundation under grants No. IRI-8802390, IRI-9000532 and by a Presidential Young Investigator award to Bruce Donald, and in part by the Air Force Office of Sponsored Research, the Mathematical Sciences Institute, Intel Corporation, and AT&T Bell Laboratories.

†Supported in part by ONR Grant N00014-88K-0591, ONR Grant N00014-89J-1946, NSF Grant DMC-86-17355 at Cornell University, and by NSERC Operating Grant OGP0122128 at UBC

of mechanics, and combinatorial collision detection algorithms. From this viewpoint, subtle mathematical difficulties arise in predicting motions under rotational compliance, such as the forced non-genericity of the intersection problems encountered in configuration space. We discuss these problems and their solutions. Finally, we extend our work to predict the forces on the manipulated objects as a function of time, and show how our algorithm can easily be extended to include uncertainty in control and initial conditions. With these extensions, we hope that our system could be used to analyze and design objects that are easy to assemble, even given control and sensing errors, and that require more force to disassemble than to mate.

Contents

1 Introduction

We are pursuing an algorithmic theory of *design for assembly*. To this end we are developing and implementing algorithms that can analyze and generate designs for objects so that they will be easy to assemble. In particular, we observe that real objects that robots might assemble are typically not rigid. For example, a Sony Walkman is made of plastic parts that snap together. Significant advances were made in the design of the IBM ProPrinter, by replacing traditional fasteners such as screws with plastic parts that simply snap together. The reason these plastic parts snap together is that they are *flexible*: more precisely, they are *passively compliant*. This means that when the parts are brought together and an external force applied, the parts deform in a prescribed way. More interestingly, the force required to mate two parts may be much less than the force required to take them apart.

Since we wish to be able to design and have our robots assemble such objects given task-level descriptions, we must have a systematic program for reasoning about and predicting their motions in contact. To this end we make precise a sufficiently powerful notion of flexibility to model the objects above, which encompasses several important and complicated mechanisms in mechanical design and automated assembly: snap-fasteners, latches, ratchet and pawl mechanisms, and escapements (see fig. 1). We model the physics of interaction of the flexible parts and the obstacles using generalized springs, quasi-static analysis and Coulomb friction. Using these tools, we can proceed to develop combinatorially precise geometric algorithms for predicting the motion of a flexible object near and in contact with its mating part. A system based on the results of this paper is also described as well as the relevance to the design of such mechanisms.

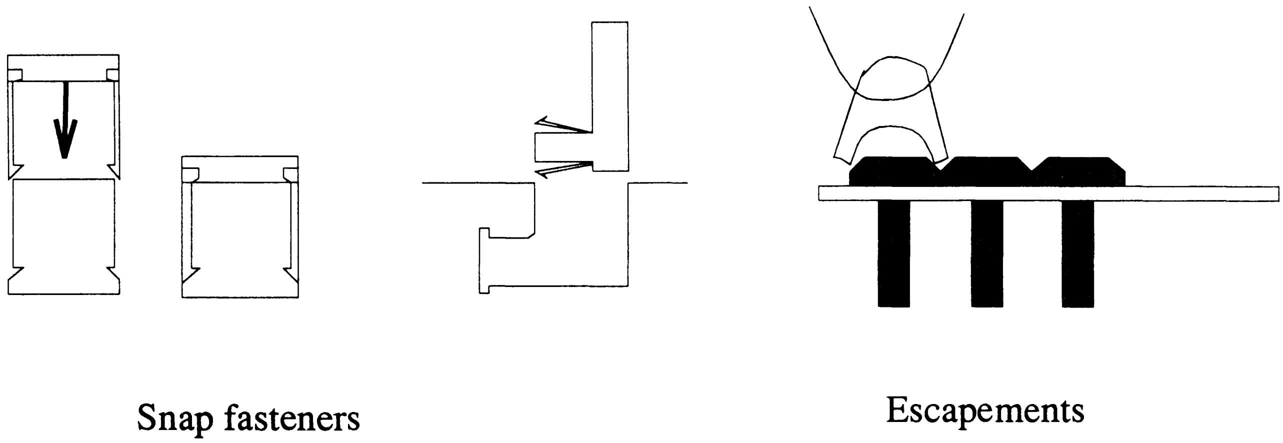


Figure 1: Examples of compliantly connected rigid bodies.

From a complexity-theoretic standpoint, our result may also be viewed as follows. Earlier work on compliant motion planning [LMT84, Erd84, Don88b] has employed a dynamic model called the *generalized damper* [Whi76]. While this model has led to exact algorithms in the case of pure translations [Don88b, Can89], so far we have not been able to provide combinatorially precise algorithms once rotations are permitted. This is true even if the commanded motion is a pure translation and the moving object (eg., the peg) is merely rotationally compliant. One reason for this difficulty seems to be that under the damper model, the resulting motions are not obviously algebraic; even more disturbing, the outer envelope (or *forward projection*) of these motions cannot yet be algebraically described. (This is also a problem in many more complicated dynamical systems). However, under the model in this paper, rotationally compliant subparts are permitted, and yet all resultant trajectories of the system are algebraic. Finally, we believe that our algorithm is sufficiently simple that it could be implemented, and used to plan and verify the design of flexible mechanisms for ease of assembly.

We have implemented our algorithm, and in this paper we describe the details of the algorithm, implementation, and experiments. We have built a system for predicting and analyzing the motion of "snap-fastener"-type devices, and we describe experiments we have run to analyze and design particular objects.

A major impediment to developing systems such as ours has been the apparent necessity to integrate out the differential mechanics in order to determine the long-term behavior of the system. This problem is exacerbated by the fact that in many models of rotational compliance such as the generalized damper (eg., [LMT, Erd, Don88a,b, Can89]), the resulting trajectories are not known to be algebraic; neither do we have ways of computing algebraic bounding approximations (or forward projections). We begin by discussing in quite general terms how our model of compliance permits us to obtain algebraic, closed-form solutions to motion prediction problems for a rotationally compliant object, and how this leads to *exact* algorithms for analyzing designs for assembly.

We continue by discussing the relevance of our approach to engineering. To this end, we give several examples of fastener-type objects. We illustrate our discussion with experiments performed with our system to automatically analyze and predict the motion during execution of the assembly plan.

Our algorithm is algebraic and exact in principle. However, unlike many theoretical algebraic algorithms, it is also implementable. A chief goal of this paper is to show how we implemented it. Next, we discuss our algorithm for motion analysis in detail. Our work is interdisciplinary, in that it is situated at the interface between differential theories of mechanics, and combinatorially precise computational approaches to collision detection and compliant motion prediction. We employ some simple tools from computational algebra, and we discuss their application in our algorithm in some detail. While these tools often seem straightforward from a theoretical viewpoint, there is a host of practical and implementational problems in trying to build a system and reduce them to practice. Many of these issues focus on the problems of robustness, non-genericity, and stability. We discuss these problems in some detail. For example, while many algebraic and computational-geometric algorithms can assume genericity (eg., general position), we can show that in the case of predicting rotational compliance, one is in effect forced to solve non-generic intersection problems in configuration space. Careful thought is required to make such algorithms robust.

Our algorithm predicts where the motion will terminate, and the configuration and contact history as functions of time. We also show how our algorithm can predict the forces experienced by the manipulated parts as a function of time. Thus, we can avoid designs that require excessive forces to assemble, and can analyze how a design can be easier to mate than to disassemble. This force-history extension requires the introduction of transcendental functions, and so it results in a numerical (not an exact) algorithm; however, any desired precision for the forces may be obtained.[1]

Previous work on algorithmic motion planning has largely concentrated on the movement problem for rigid bodies [LoP,Don,Yap]. Recently, work in compliant motion planning under uncertainty [LMT, Erd, CR, Don2, Don3, Can2, Bri, FHS, Bro, Bro2] has focused on the problem of moving rigid objects (e.g. pegs) in contact (i.e., compliantly) with obstacles (such as holes) under force-control, subject to bounded uncertainty and error. The only combinatorially precise results that have been obtained for compliant motion have been for pure translations. Hence they are essentially inapplicable for any real systems, which typically can rotate. Indeed, planning and simulation for systems with rotational bodies and rotational compliance has resisted solution in the sense that only approximate, heuristic, or numerical algorithms are known. We consider a mechanical system of bodies that can compliantly slide on obstacles while they translate and rotate in the plane. We provide an exact, efficient algorithm for predicting the motion of these rather interesting devices, which, as we we discuss below, are important in manufacturing, design, and factory assembly. The key ideas we use are: red-blue merge algorithms, a simple dynamical systems model, and local dynamic constraints. These tools permit us to reduce the simulation to a plane sweep of a " dynamically annotated" slice of configuration space. We hope that the paradigm of "simulation as sweep" may be useful in other domains.

We view the simulation problem, even with rotational compliance and quasi-static mechanics, as a problem that can be solved by careful reduction to a plane sweep. In particular, for each "pawl," we reduce to sweeping an planar arrangement of algebraic curves of low degree. The connected component of free space defined in this planar arrangement has complexity $O(\lambda_r(n))$, and can be constructed in time $O(\lambda_r(n)\log^2 n)$ using a red-blue merge algorithm [GSS]. Here $\lambda_r(n)$ is the (almost linear) maximum length of (n,r) Davenport Schinzel sequences [GSS]. r is a small constant related to the number of times two cubic[2] configuration space constraint curves can intersect. Our approach to modeling rotational compliance and to incorporating frictional constraints leads to the first formulation of the simulation prediction problem which permits a *reduction* of motion prediction to plane sweep. Our solution differs from previous work on predicting, bounding, and planning rotationally compliant motions with quasi-static mechanics in that it is (i) *purely algebraic*, and hence exact, (ii) *combinatorially precise*, in that the computational complexity is exactly known, and (iii) requires no integration.[3]

A preliminary version of this paper appeared in greatly abbreviated form in the *Proc. IEEE Int. Conf. on Robotics and Automation* in 1990.

PART I: OVERVIEW

[1]By substituting a polynomial approximation for the transcendental functions, the algorithm could be made an ϵ-approximation scheme.

[2]By "cubic" we mean the total degree of the defining multinomial is 3. Our curves have additional structure, such as low-degree parameterizations, as well.

[3]Note that [Don2,Can2,Bri,FHS] address geometric reachability issues for translationally compliant objects, but these objects cannot rotate.

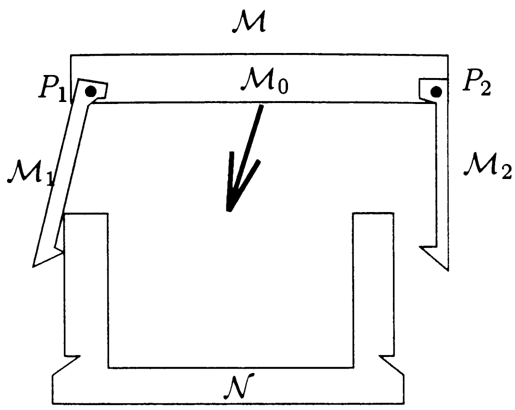


Figure 2: Linked body $\mathcal{M}$ moving among $\mathcal{N}$.

2 Problem Statement

We consider the problem of moving a flexible, linked body $\mathcal{M}$ in the plane, in a polygonal environment $\mathcal{N}$. The flexible body $\mathcal{M}$ is modelled as follows: polygons $\mathcal{M}_h$, $h = 1, \ldots, k$, called "pawls", are attached to a root polygon $\mathcal{M}_0$, at hinge points P_h. Each hinge is coupled with a spring of stiffness κ_h. (See Figure 2). We refer to the static environment, $\mathcal{N}$, as "obstacles", although in fact, it will probably consist of fixtures and/or the mating half of the manipulated part.

The motion of the body $\mathcal{M}$ consists of rigid translation of the root polygon $\mathcal{M}_0$. The pawls $\mathcal{M}_h$ are free to move compliantly as dictated by interactions with the environment and the spring.

Time T will be represented by the nonnegative real numbers. A *solution trajectory* Φ for the system $(\mathcal{M},\mathcal{N})$ is a family of maps $(\phi_0, \phi_1, \ldots, \phi_k)$ where $\phi_0 : T \to \Re^2$ specifies the configuration of the root and each $\phi_h : T \to S^1$ specifies the orientation of pawl polygon $\mathcal{M}_h$, $h = 1, \ldots, k$. Hence the global configuration of pawl $\mathcal{M}_h$ at time t is given by $(\boldsymbol{p}(t), \phi_h(t))$, where $\boldsymbol{p}(t)$ differs by a constant offset from $\phi_0(t)$. We see that the configuration space of the system $(\mathcal{M},\mathcal{N})$ is[4] $C = \Re^2 \times \overbrace{S^1 \times \cdots \times S^1}^{k\ S^1\text{'s}}$.

Definition 2.1 *The* Simulation Problem *is to determine:*

1. *The solution trajectory $\Phi : T \to C$ of the system.*
2. *The time of termination of the motion, the cause of termination (such as sticking due to friction, sticking due to kinematic constraints, etc.), and the configuration of $\mathcal{M}$ at termination.*

[4]This is the configuration space with no springs, but only kinematic constraints. Introduction of springs means that it is not possible to identify a rotation of 2π with 0, since at 2π the pawl is "cocked". Hence the introduction of dynamics forces us to pass to the covering space $\Re^{k+2}$. Our analysis goes through *mutatis mutandis* for the covering space.

3. *The time history of contacts between $\mathcal{M}$ and $\mathcal{N}$.*

We also address some extensions of the simulation problem in Appendices A,B including the determination of the time history of forces during the motion and the effect of uncertainty.

We make the following assumptions about the physics of object interactions and the motion of $\mathcal{M}$:

- Object interactions are restricted to those between $\mathcal{M}_h$ and $\mathcal{N}$. In other words, pawls do not collide with each other, but may collide with the environment $\mathcal{N}$. The effect of this assumption is to make the motion of each pawl independent of the motion of other pawls. Henceforth we shall consider the motion of a single pawl $\mathcal{M}_h$.

- Since the root $\mathcal{M}_0$ is undergoing a rigid translation, so does the hinge point P_h. We shall assume that this translation is a straight line motion given by

$$\boldsymbol{p}(t) = \boldsymbol{p}_o + \dot{\boldsymbol{p}}t \tag{1}$$

  where t is the time, $\boldsymbol{p}_o$ is the initial position (at $t = 0$) of P_h, and $\dot{\boldsymbol{p}}$ $(= \dot{\phi}_0)$ is its velocity.

- Stable contact: Suppose the pawl is in contact with a feature (edge or vertex) of the environment (for example, during sliding). We assume that if we perform a small displacement of the pawl away from the environment, the torque on the pawl due to the spring is such that the contact will be restored. This assumption is not very restrictive at all – in fact, in the face of even the smallest uncertainty, stable contacts are the only ones one can hope to observe in practice.

- Quasi-static motion: The motion is assumed to be slow enough that inertial effects are not significant. This corresponds to assuming that there is no acceleration of the pawl, and hence the forces on the pawl are balanced. The quasi-static assumption is reasonable at small speeds and is widely used (see, for example, [Whi82, Mas82, Don88a, Pai88]).

- If the pawl slides off $\mathcal{N}$ into free space, it may have a residual torque due to the spring being cocked. We shall assume that the pawl rotates back towards its rest orientation at such great speed that $\boldsymbol{p}$ does not change significantly during the rotation and can be taken to be constant. This, incidentally, is the "snap" in the snap-fasteners that we wish to model. This assumption may appear to contradict the assumption of quasi-static motion, but is in fact practically a consequence. Quasi-static motion implies that the root is moving "slowly-enough" for the forces to be balanced. Hence, when a pawl is in free space and has a residual torque, its motion can be fast compared to that of the root, resulting in a "snap". This assumption can be relaxed by assuming a linear relationship between the translation $\boldsymbol{p}$ and the rotation (e.g., see [Can86]), but we do not deal with it here.

- The forces of friction arising from contact obey Coulomb's Law. We further assume that there is a single coefficient of friction. This assumption is also widely used.

Figure 3: The root body is rectangular, and the motion plan is straight down. The two pawls are "hook-shaped", and are attached with torsional springs at the hinges to the root body. The environment ("obstacles") are the two filled-in rectangles meeting at right angles; they form a "T"-shaped black body. In this example, it seems that the pawls will clearly comply to the "T" and snap around it; presumably this device cannot be disassembled by reversing the assembly plan. In fact, our algorithm shows that (i) the right-hand pawl cannot reach the base of the "T", and (ii) with low friction, there is no obstacle to disassembling the device.

These assumptions define a simple but adequate dynamical system. We will exploit the geometry of this system extensively to obtain our results.

3 Examples

3.1 The Two-Pawl Example

Figure 3 shows a flexible device which is to move and snap around a mating part (a black, "T"-shaped object). Now, for a real device, the pawls will be tiny in comparison to the root body and mating parts, but we have made them very large here so as to illustrate the geometric interaction. In this example, the coefficient of friction is very small. This enables the reverse motion to "pull the pawl back up" out of its mating part.

The "assembly plan" for the parts consists of a straight-line translation of the root body; this motion is parameterized by time and is specified by an initial configuration and a velocity vector. (Throughout this paper, we will use the term "assembly/disassembly plan" to mean such a straight-line translation). As the pawls contact the environment, they can move in a prescribed way from the contact forces. In particular, each pawl is attached to the root by a torsional spring. The pawls can also "snap" off the obstacle edge back to their zero position; this motion is modeled using a pure rotation. As the motion proceeds, the pawls deform (deflect) around the obstacles in response to the kinematic constraints and the contact forces. See fig 4.

When a root collision is detected, the system attempts to reverse the assembly plan and pull straight up. Because there is no friction, the pawl can be removed. If the coefficient of friction is increased, then the left pawl will stick on left of the "T" environment, and the flexible object cannot be pulled back up. This example shows that while it is relatively clear, intuitively, that during assembly the pawls may snap around the black "T"-shaped object, that even this simple problem holds some surprises. First, the right-hand pawl cannot actually reach around the base of the "T" before the root collision occurs. Second, it is not *a priori* clear that the left pawl will not stick during the reverse (disassembly) plan. In fact, it will not stick on the left of the "T" merely due to kinematic constraints; friction is required.

Fig. 4 was generated by our algorithm. Kinematic constraints are modeled using configuration space C-surfaces, as in [LoP, Don87, Bro2]. Reaction forces are modeled using Coulomb friction. The dynamics of the system are assumed to be quasi-static [Whi82, Mas86]; we regard this assumption as a 0^{th}-order approximation of the real dynamical system.

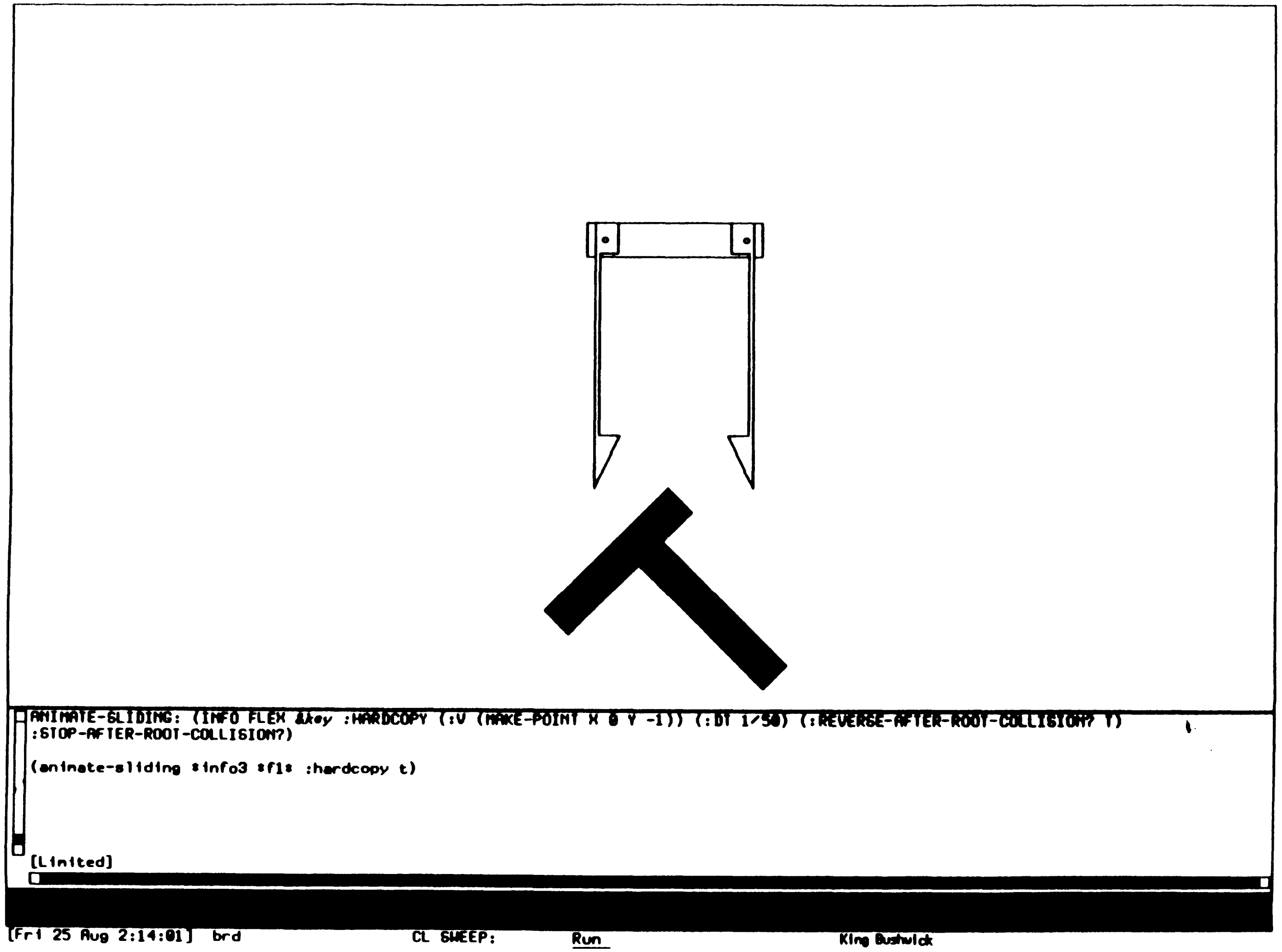

ANIMATE-SLIDING: (INFO FLEX &key :HARDCOPY (:V (MAKE-POINT X 0 Y -1)) (:DT 1/50) (:REVERSE-AFTER-ROOT-COLLISION? T)
:STOP-AFTER-ROOT-COLLISION?)
(animate-sliding *info3 *f1* :hardcopy t)
[Linited]
[Fri 25 Aug 2:14:01] brd
CL SWEEP:
Run
King Bushwick

Figure 3

Figure 4: The result of running our algorithm on the two-pawl example. Notice how the pawls comply around the obstacles as they deform in response to the kinematic constraints and the contact forces. Figs. 4a-g show the motion downwards. At 4g, the root collides with the "T", and the motion is reversed. Figs. 4h-o show the reverse motion upwards.

Figure 5: "Motion diode" example. There is no root body, and only one triangular pawl which can pivot about a torsional spring at its center. Pure translations of the diode can be commanded, and the triangular pawl deflects in response to contact forces from the environment. These figures were generated by running our analysis algorithm on the data shown. In 5c, the pawl snaps off the upper right obstacle and continues downward. Fig. 6 shows the reverse motion, during which the pawl gets stuck comping back up.

3.2 "Motion Diode" Example

In design for assembly, we often desire "locking" parts that, when mated, cannot be disassembled by motion plans in a particular family of directions. More generally, we may require interlocking parts that cannot be disassembled at all, for any translational motion plan. Most generally, one might want parts that cannot be disassembled without exerting large forces (see sec. A). Following a suggestion of Mason [Ma84], we call such objects "motion diodes." The term is motivated by the fact that motion is possible in certain directions, but not in others. Our usage differs from Mason's, in that his motion diodes are geometries from which a robot cannot be *guaranteed* to emerge. Our motion diodes are (flexible object, environment) pairs such that for some family of controls, (or perhaps all controls), no change in the sign of the controls can reverse the motion to reachieve the start position. In our simple case a "plan" is a translation given as a straight line motion

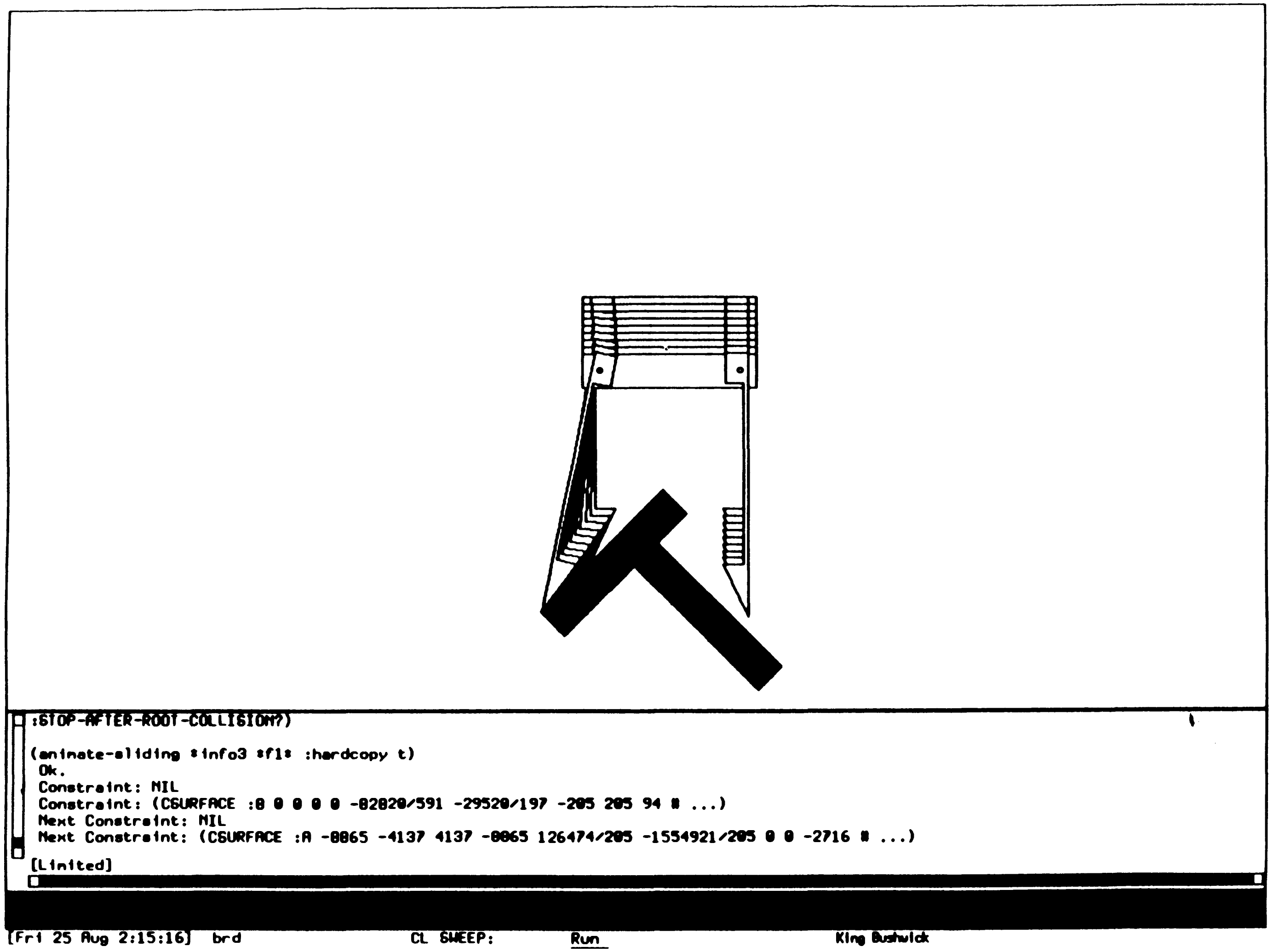
$$\boldsymbol{p} = \boldsymbol{p}_o + \dot{\boldsymbol{p}}t \tag{2}$$

where t is the time, $\boldsymbol{p}_o$ is the initial position (at $t = 0$), and $\dot{\boldsymbol{p}}$ is the root velocity. A family of controls corresponds to a set of velocities $\{\dot{\boldsymbol{p}}\}$, and changing the sign amounts to specifying $-\dot{\boldsymbol{p}}$.

If motion diodes can be designed, analyzed and verified, then they can be rigidly attached as "fasteners" to bodies that we wish to mate, but not to disassemble. For example, if the triangular pawl in fig. 5 is attached in the z-axis (perpendicular to the figure) to a root body in a parallel x-y plane to the figure, then the root body can be fastened irreversibly to its mating part.

Our algorithm can analyze designs for these kinds of diodes. In fig. 4, the two-pawl device and the T-shaped object would form a motion diode with respect to pure translation in y, for sufficiently high coefficients of friction.

Now see fig. 5. In this example, there is no root body. The one triangular pawl can pivot about its center; a torsional spring is attached at the pivot. The pawl is moved down in a pure $-y$ translation, and in response to the reaction forces from the environment, it rotates compliantly. Let us label the black obstacles, starting with the uppermost one, in clockwise order, A, B, and C. The pawl contacts A and rotates counterclockwise while



:STOP-AFTER-ROOT-COLLISION?)
(animate-sliding *info3 *f1* :hardcopy t)
Ok.
Constraint: NIL
Constraint: (CSURFACE :0 0 0 0 0 -82820/591 -29520/197 -205 205 94 # ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -8865 -4137 4137 -8865 126474/205 -1554921/205 0 0 -2716 # ...)
[Limited]
[Fri 25 Aug 2:15:16] brd
CL SWEEP:
Run
King Bushwick

Figure 4a

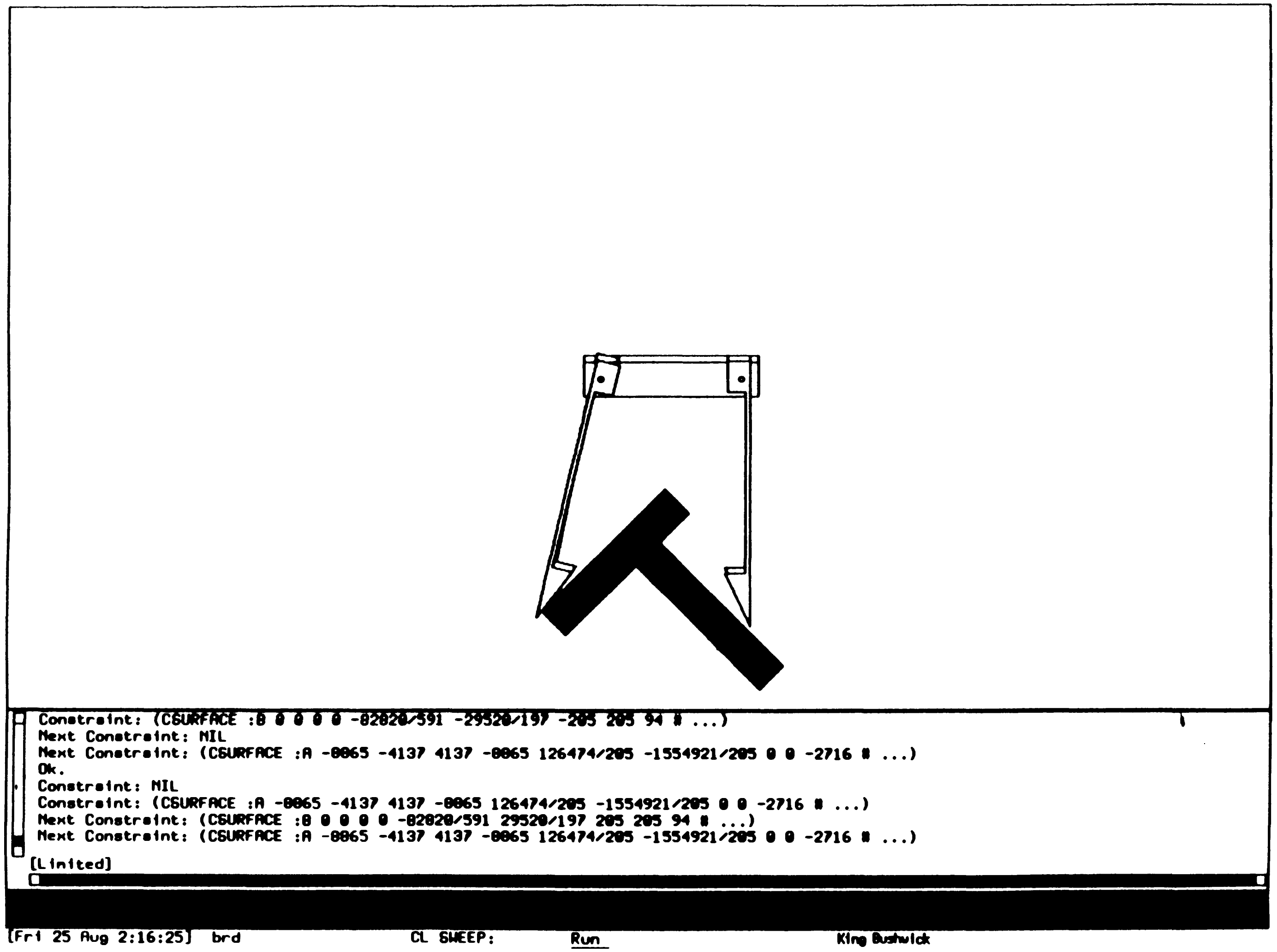

Constraint: (CSURFACE :B 0 0 0 0 -82820/591 -29520/197 -205 205 94 # ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 # ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -82820/591 29520/197 205 205 94 # ...)
Next Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 # ...)
[Linited]
[Fri 25 Aug 2:16:25] brd
CL SWEEP:
Run
King Bushwick

Figure 4b

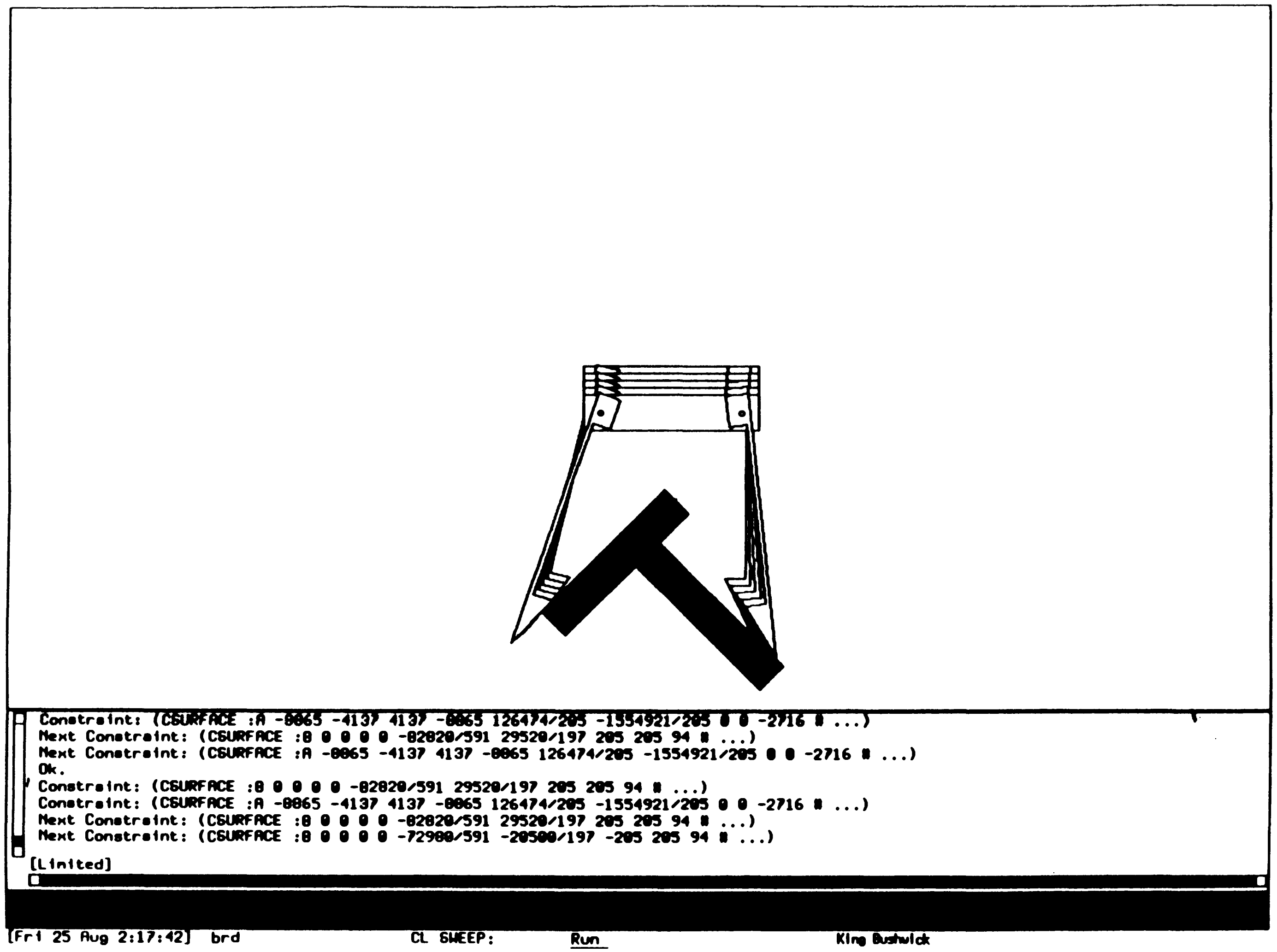
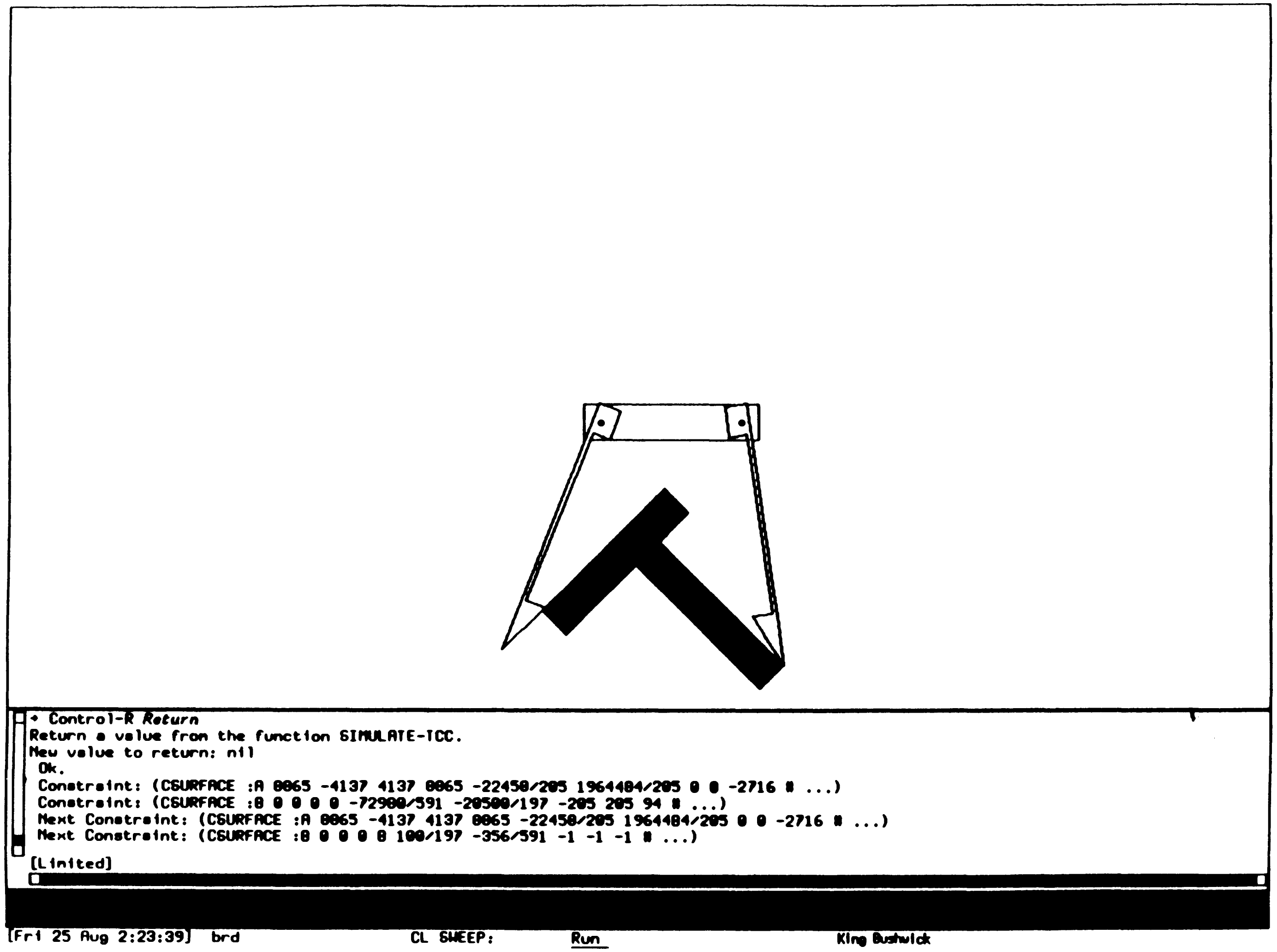

Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -82820/591 29520/197 205 205 94 # ...)
Next Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 # ...)
Ok.
Constraint: (CSURFACE :B 0 0 0 0 -82820/591 29520/197 205 205 94 # ...)
Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -82820/591 29520/197 205 205 94 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -72980/591 -20500/197 -205 205 94 # ...)
[Limited]
[Fri 25 Aug 2:17:42] brd
CL SWEEP:
Run
King Bushwick

Figure 4c



→ Control-R Return
Return a value from the function SIMULATE-TCC.
New value to return: nil
Ok.
Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 # ...)
Constraint: (CSURFACE :B 0 0 0 0 -72900/591 -20500/197 -205 205 94 # ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 # ...)
[Limited]
[Fri 25 Aug 2:23:39] brd
CL SWEEP:
Run
King Bushwick

Figure 4d

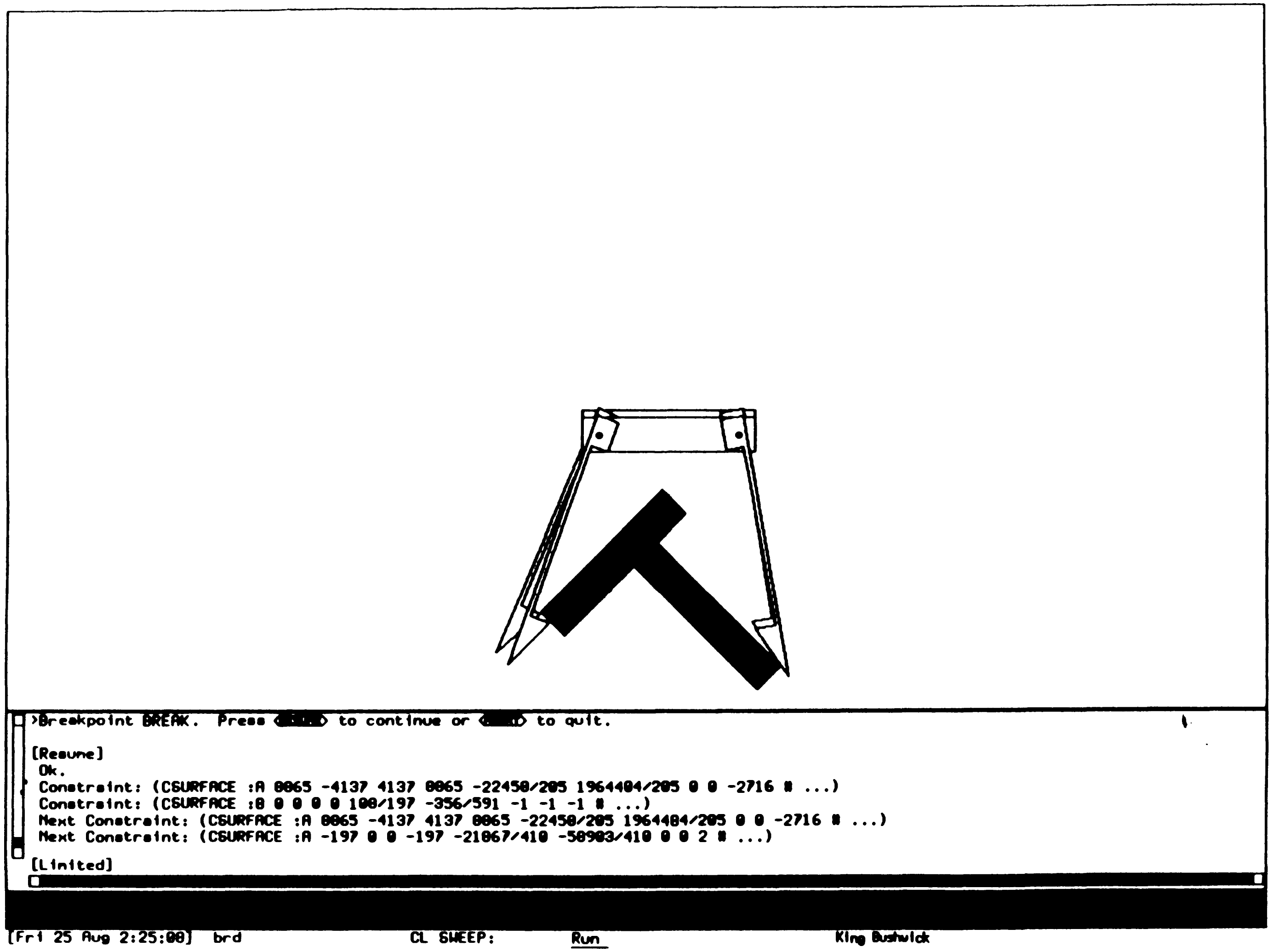

>Breakpoint BREAK. Press to continue or to quit.
[Resume]
Ok.
Constraint: (CSURFACE :A 0065 -4137 4137 0065 -22450/205 1964404/205 0 0 -2716 # ...)
Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 # ...)
Next Constraint: (CSURFACE :A 0065 -4137 4137 0065 -22450/205 1964404/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -50903/410 0 0 2 # ...)
[Limited]
[Fri 25 Aug 2:25:00] brd
CL SWEEP:
Run
King Bushwick

Figure 4e

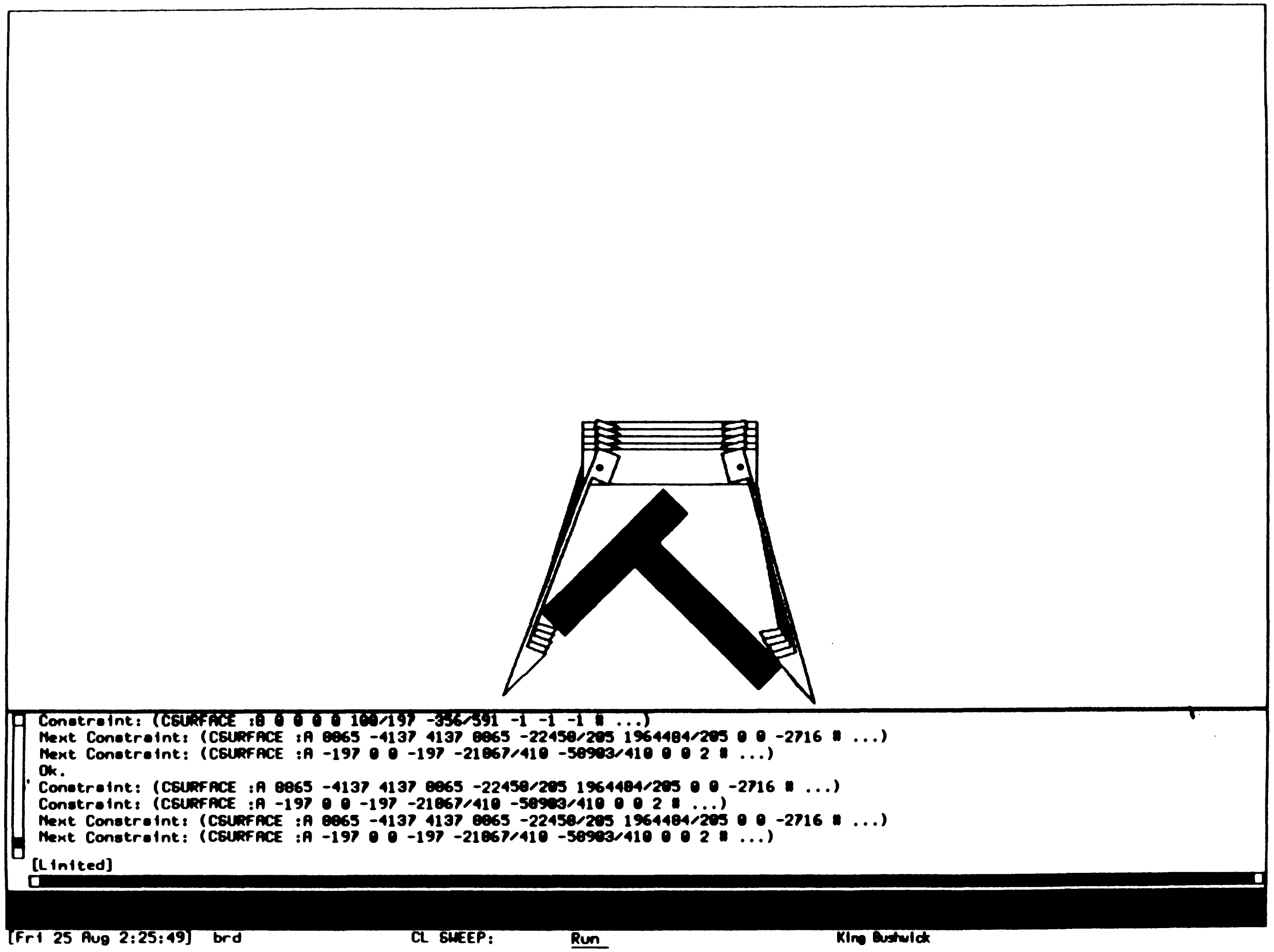
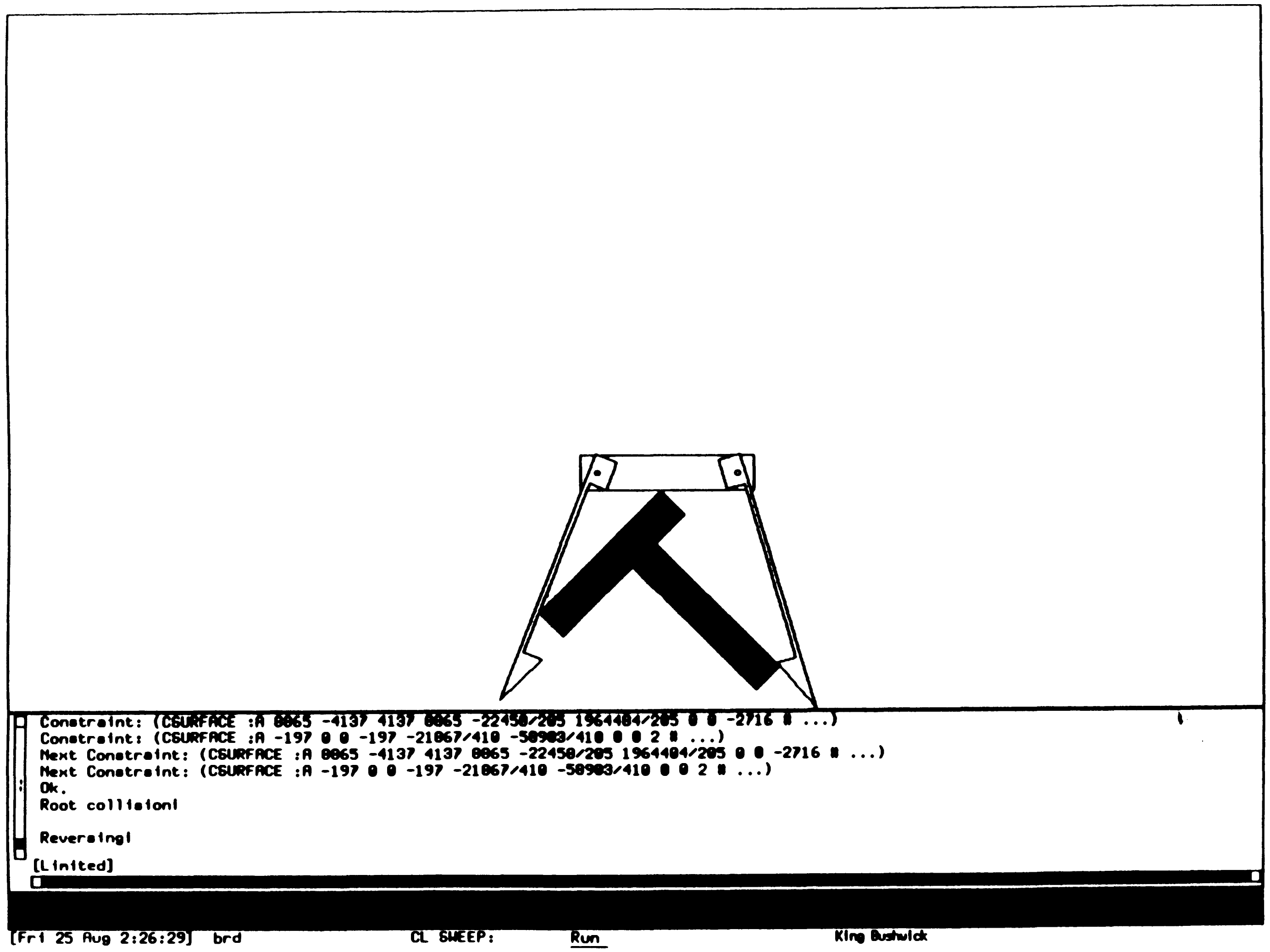

Constraint: (CSURFACE :B 0 0 0 0 0 100/197 -356/591 -1 -1 -1 # ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964404/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 # ...)
Ok.
Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964404/205 0 0 -2716 # ...)
Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 # ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964404/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 # ...)
[Linited]
[Fri 25 Aug 2:25:49] brd
CL SWEEP:
Run
King Bushwick

Figure 4f



Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 # ...)
Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58983/410 0 0 2 # ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58983/410 0 0 2 # ...)
Ok.
Root collision!
Reversing!
[Limited]
[Fri 25 Aug 2:26:29] brd
CL SWEEP:
Run
King Bushwick

Figure 4g

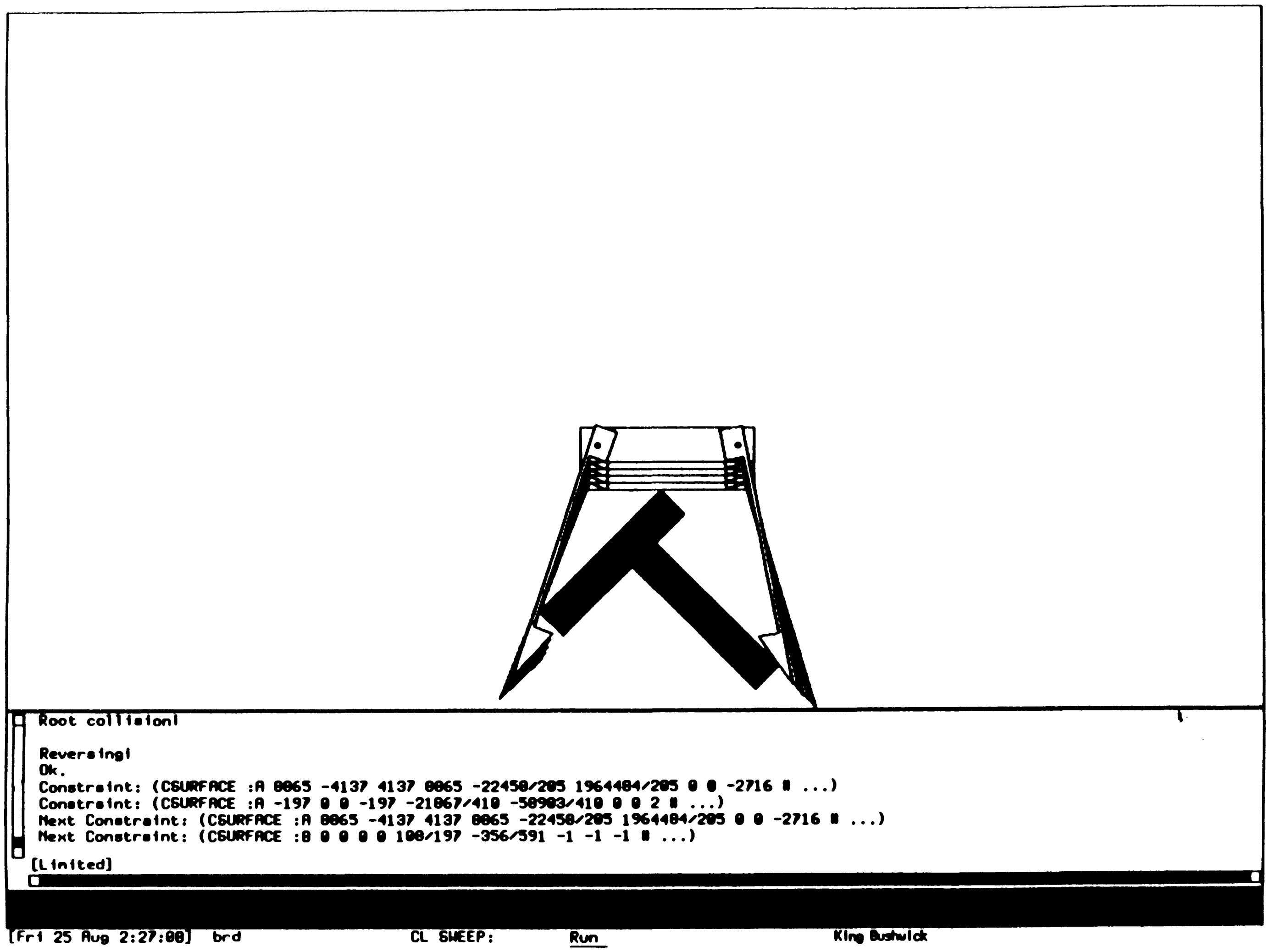

Root collision!
Reversing!
Ok.
Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964484/205 0 0 -2716 # ...)
Constraint: (CSURFACE :A -197 0 0 -197 -21067/410 -58903/410 0 0 2 # ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964484/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 # ...)
[Limited]
[Fri 25 Aug 2:27:00] brd
CL SWEEP:
Run
King Bushwick

Figure 4h

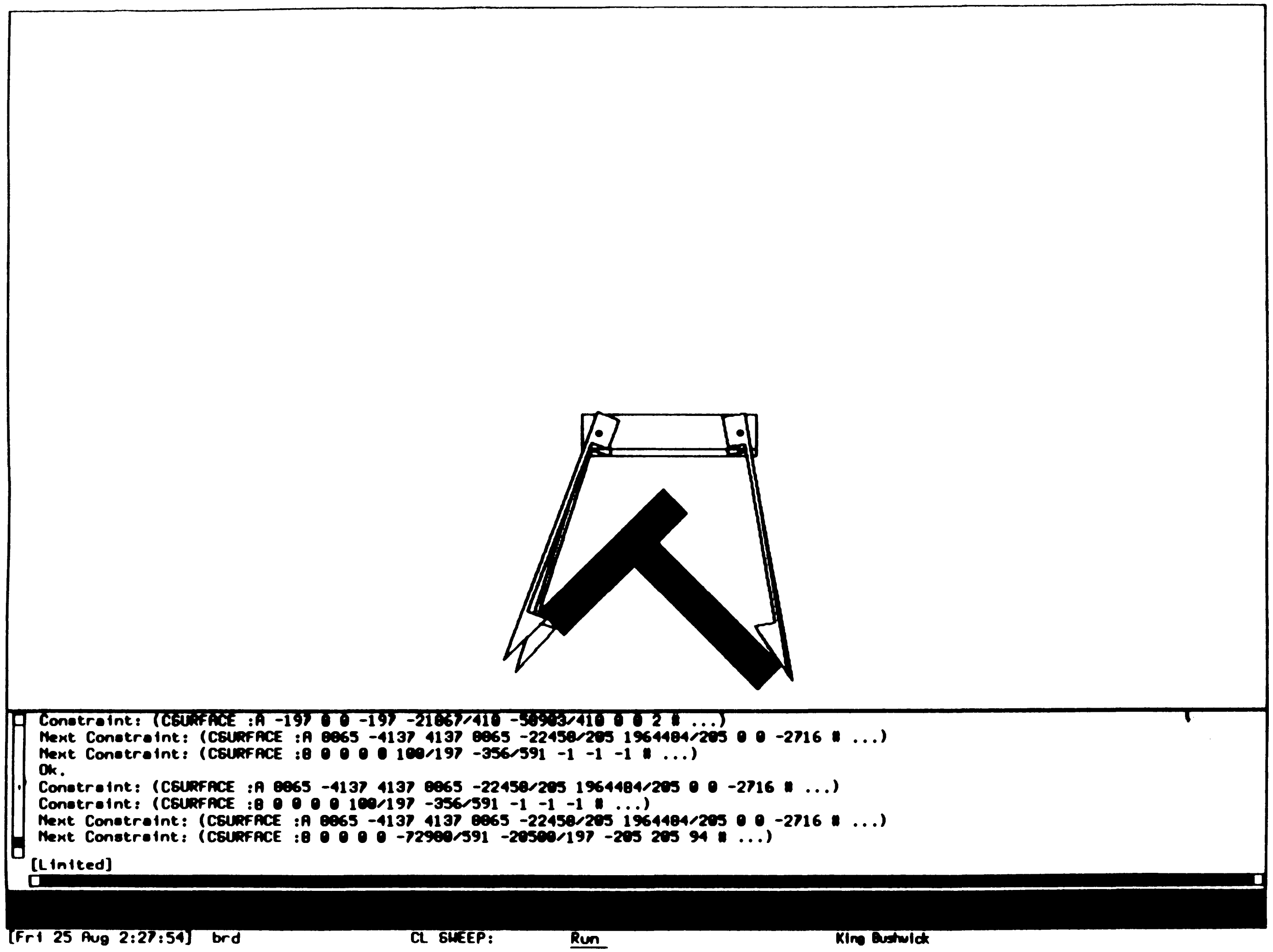

Constraint: (CSURFACE :A -197 0 0 -197 -21867/410 -50903/410 0 0 2 # ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 # ...)
Ok.
Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 # ...)
Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 # ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22450/205 1964484/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -72900/591 -20500/197 -205 205 94 # ...)
[Limited]
[Fri 25 Aug 2:27:54] brd
CL SWEEP:
Run
King Bushwick

Figure 4i

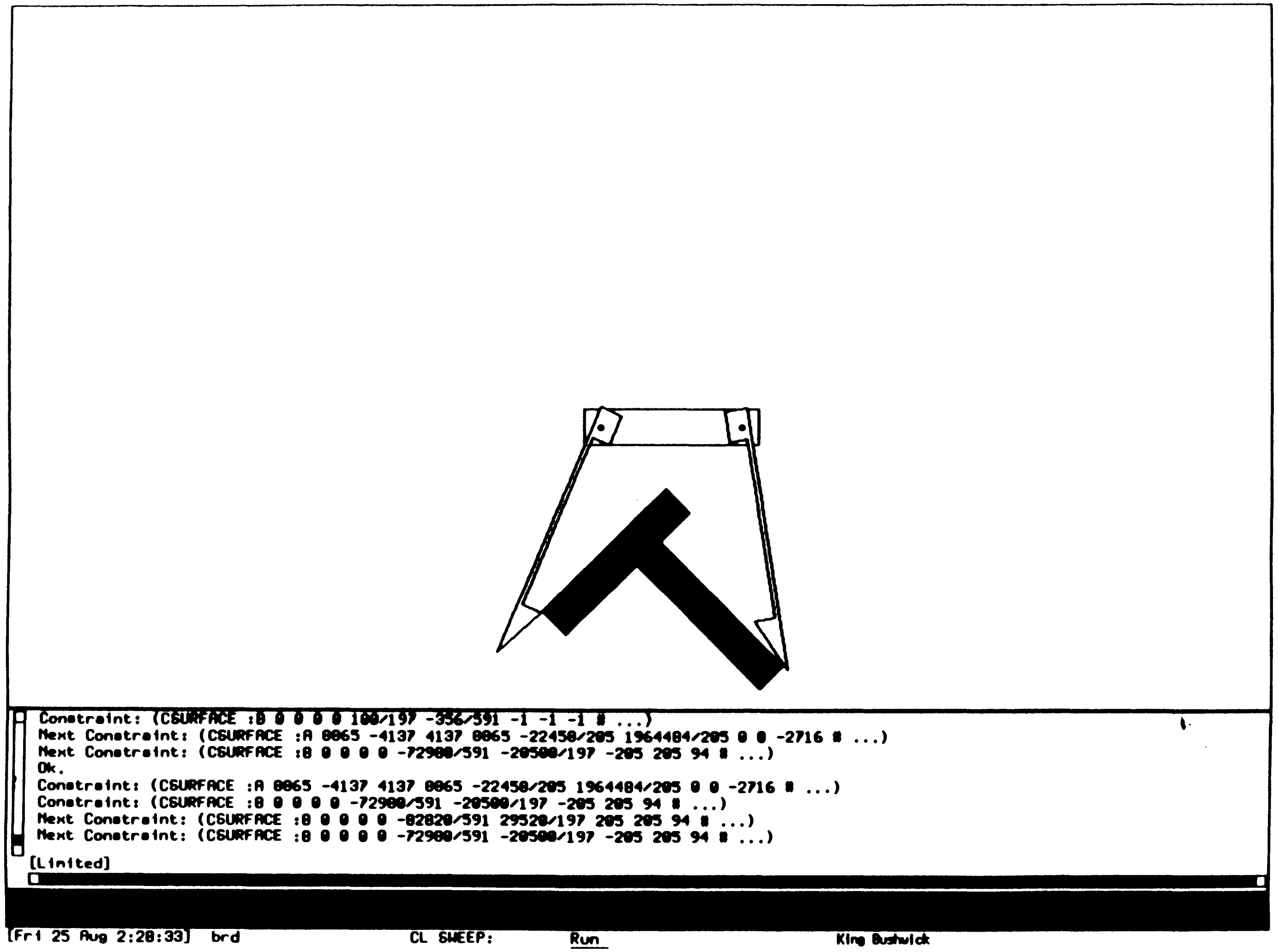

Constraint: (CSURFACE :B 0 0 0 0 100/197 -356/591 -1 -1 -1 # ...)
Next Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964484/205 0 0 -2716 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -72980/591 -20500/197 -205 205 94 # ...)
Ok.
Constraint: (CSURFACE :A 8065 -4137 4137 8065 -22458/205 1964484/205 0 0 -2716 # ...)
Constraint: (CSURFACE :B 0 0 0 0 -72980/591 -20500/197 -205 205 94 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -82820/591 29520/197 205 205 94 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -72980/591 -20500/197 -205 205 94 # ...)
[Limited]
[Fri 25 Aug 2:28:33] brd
CL SWEEP:
Run
King Bushwick

Figure 4j

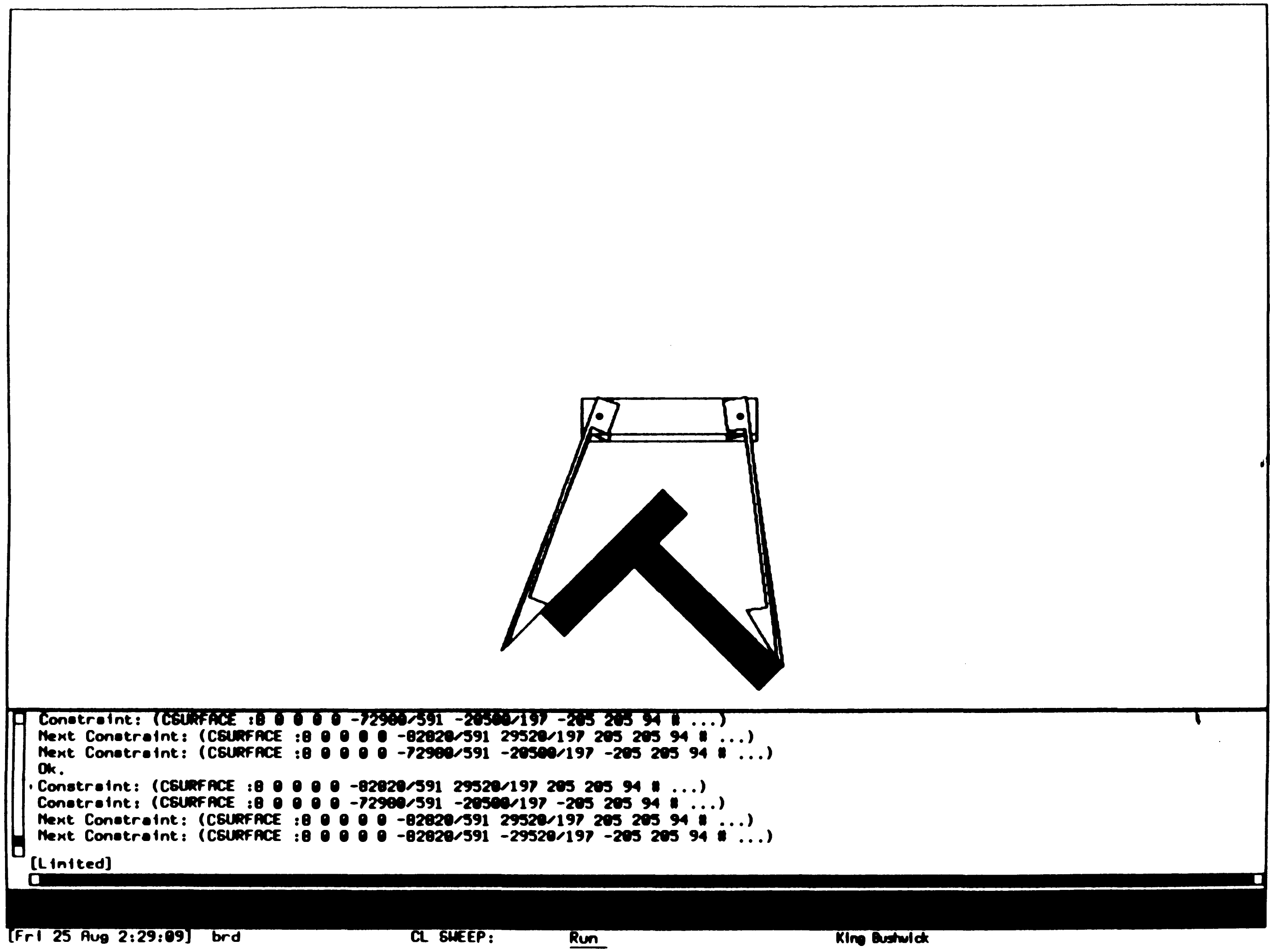

Constraint: (CSURFACE :0 0 0 0 0 -72900/591 -20500/197 -205 205 94 # ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 29520/197 205 205 94 # ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -72900/591 -20500/197 -205 205 94 # ...)
Ok.
Constraint: (CSURFACE :0 0 0 0 0 -82020/591 29520/197 205 205 94 # ...)
Constraint: (CSURFACE :0 0 0 0 0 -72900/591 -20500/197 -205 205 94 # ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 29520/197 205 205 94 # ...)
Next Constraint: (CSURFACE :0 0 0 0 0 -82020/591 -29520/197 -205 205 94 # ...)
[Limited]
[Fri 25 Aug 2:29:09] brd
CL SWEEP:
Run
King Bushwick

Figure 4k

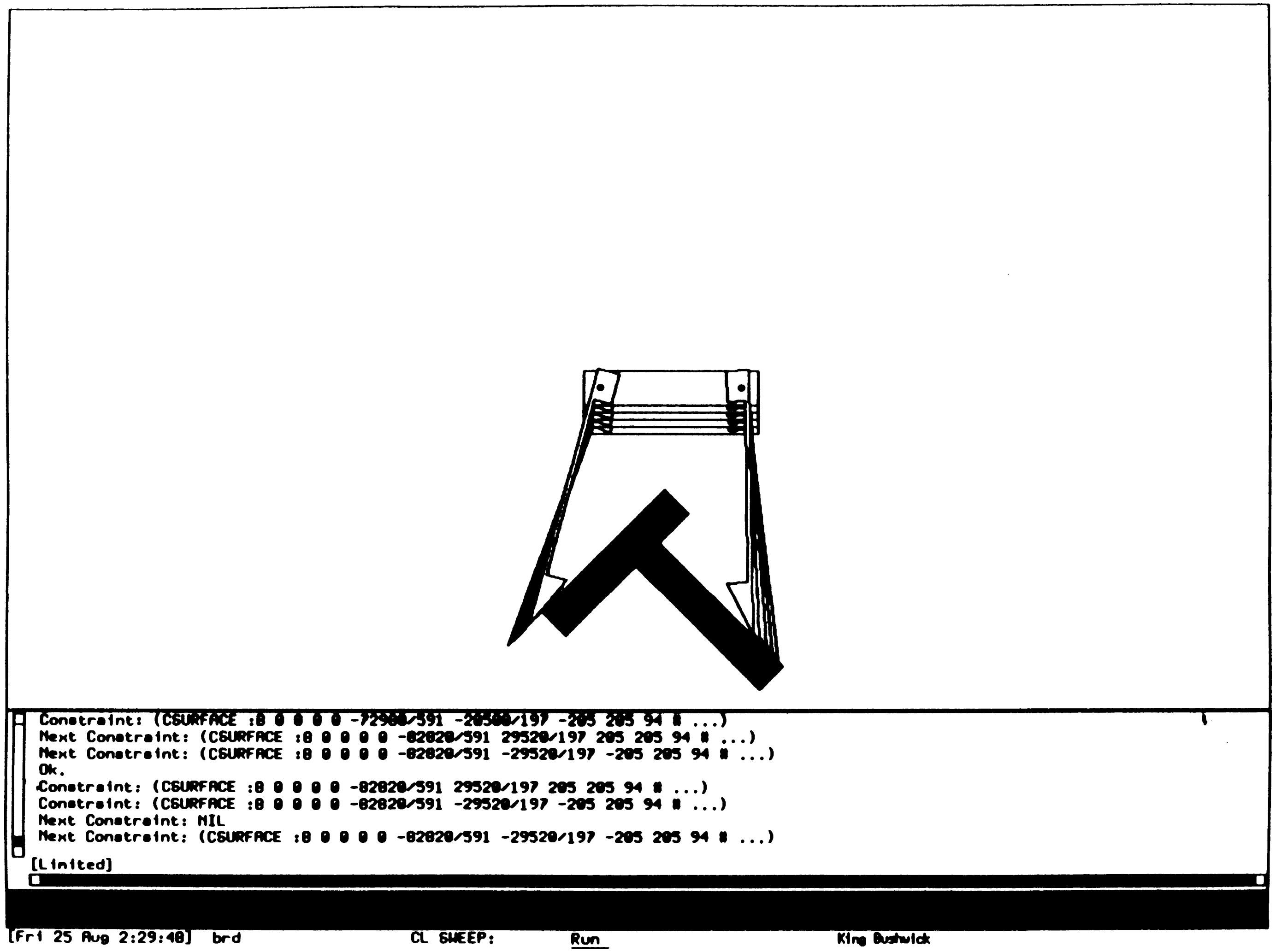
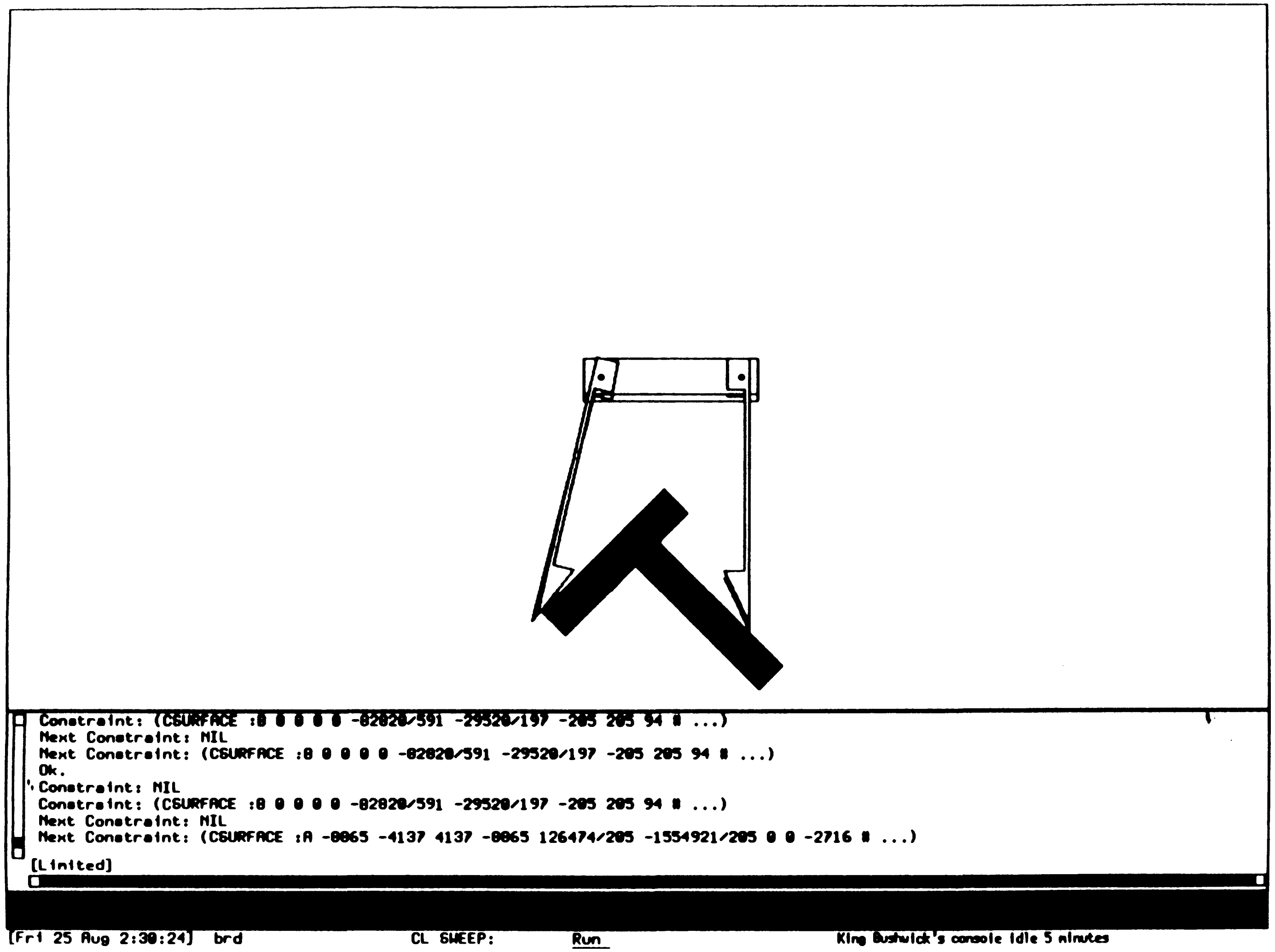

Constraint: (CSURFACE :8 0 0 0 0 -72900/591 -20500/197 -205 205 94 # ...)
Next Constraint: (CSURFACE :8 0 0 0 0 -82020/591 29520/197 205 205 94 # ...)
Next Constraint: (CSURFACE :8 0 0 0 0 -82020/591 -29520/197 -205 205 94 # ...)
Ok.
Constraint: (CSURFACE :8 0 0 0 0 -82020/591 29520/197 205 205 94 # ...)
Constraint: (CSURFACE :8 0 0 0 0 -82020/591 -29520/197 -205 205 94 # ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :8 0 0 0 0 -82020/591 -29520/197 -205 205 94 # ...)
[Linited]
[Fri 25 Aug 2:29:40] brd
CL SWEEP:
Run
King Bushwick

Figure 41



Constraint: (CSURFACE :0 0 0 0 0 -02820/591 -29520/197 -205 205 94 # ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 0 -02820/591 -29520/197 -205 205 94 # ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :0 0 0 0 0 -02820/591 -29520/197 -205 205 94 # ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -0065 -4137 4137 -0065 126474/205 -1554921/205 0 0 -2716 # ...)
[Limited]
[Fri 25 Aug 2:30:24] brd
CL SWEEP:
Run
King Bushwick's console idle 5 minutes

Figure 4m

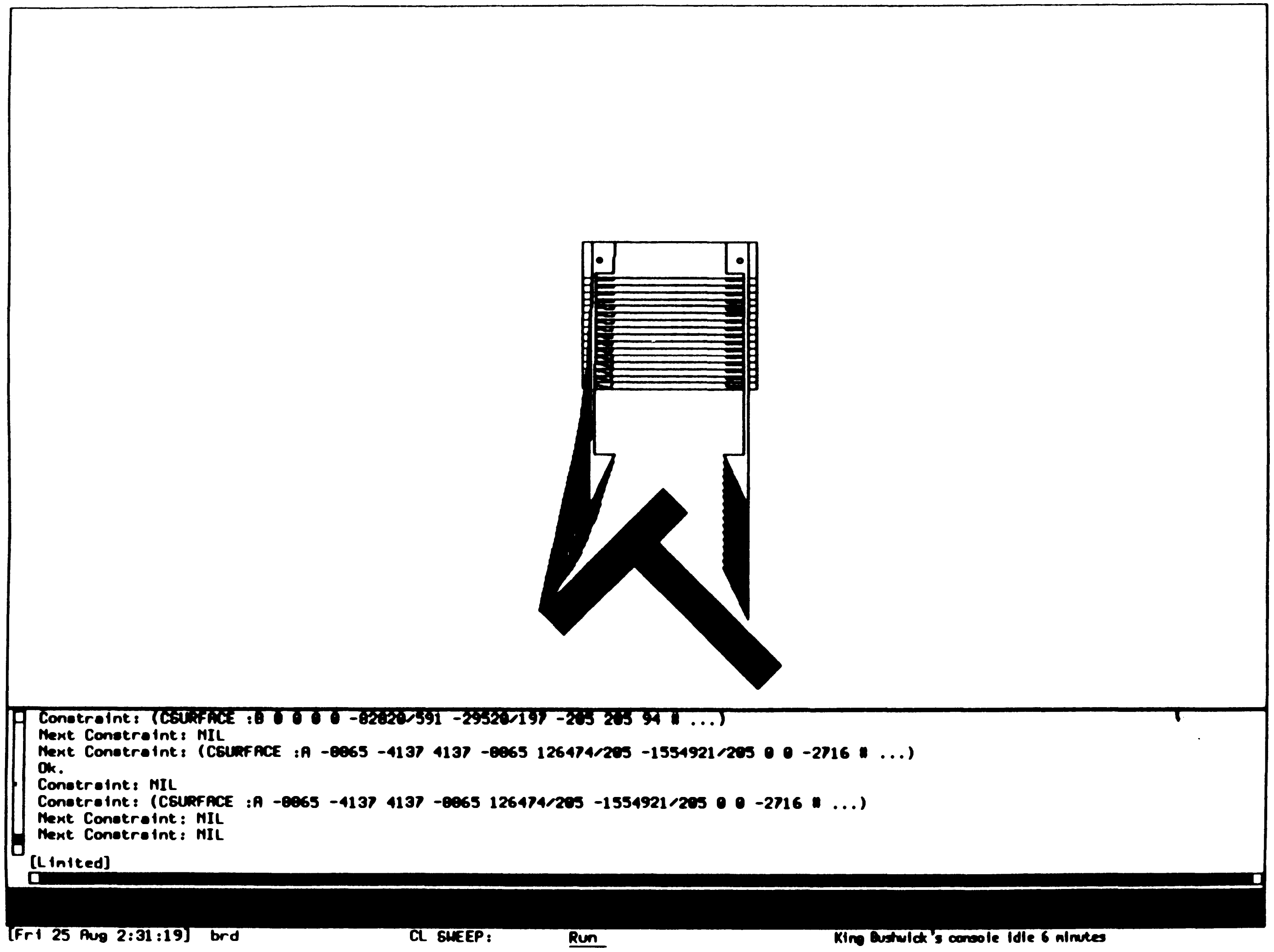
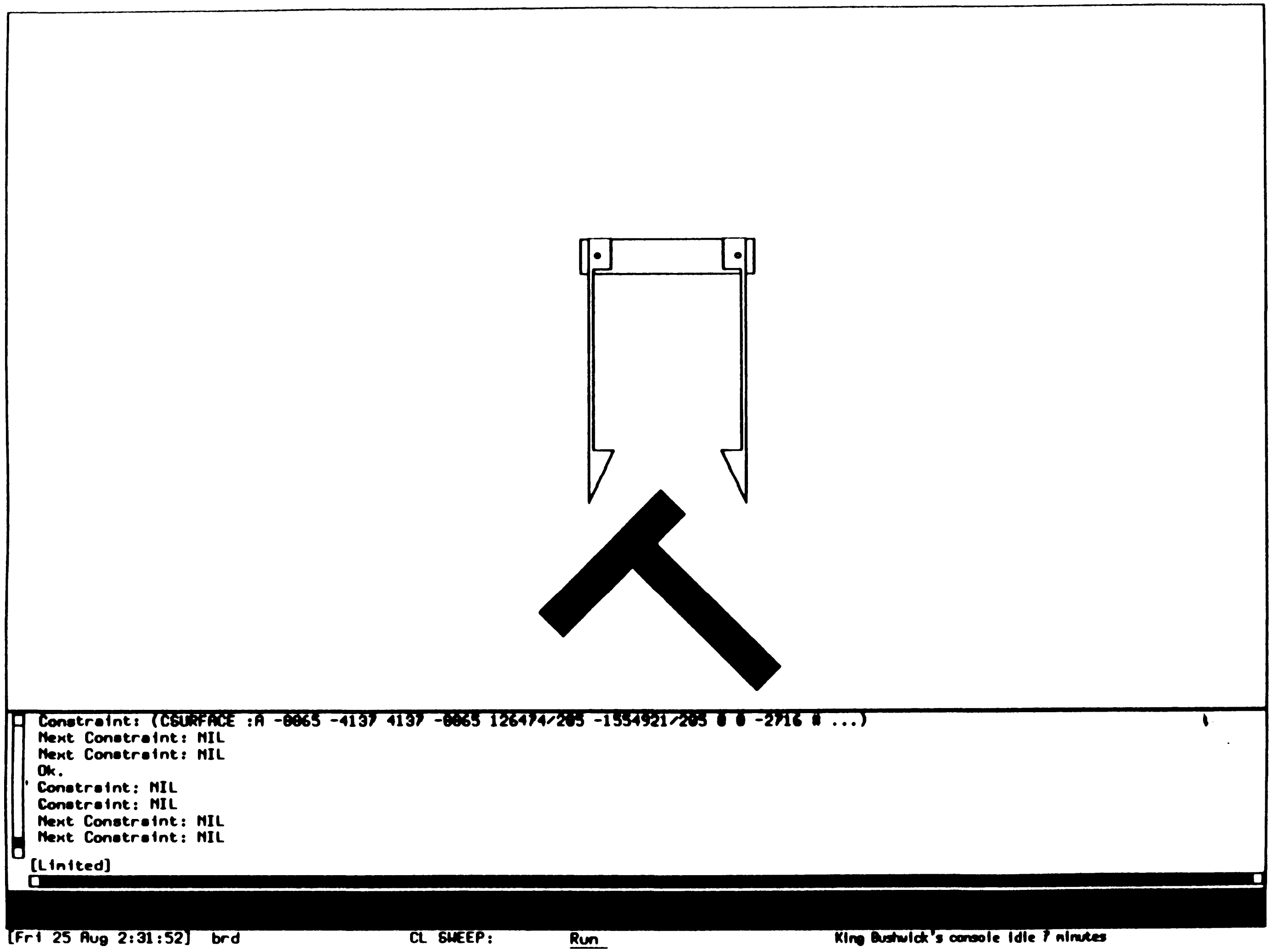
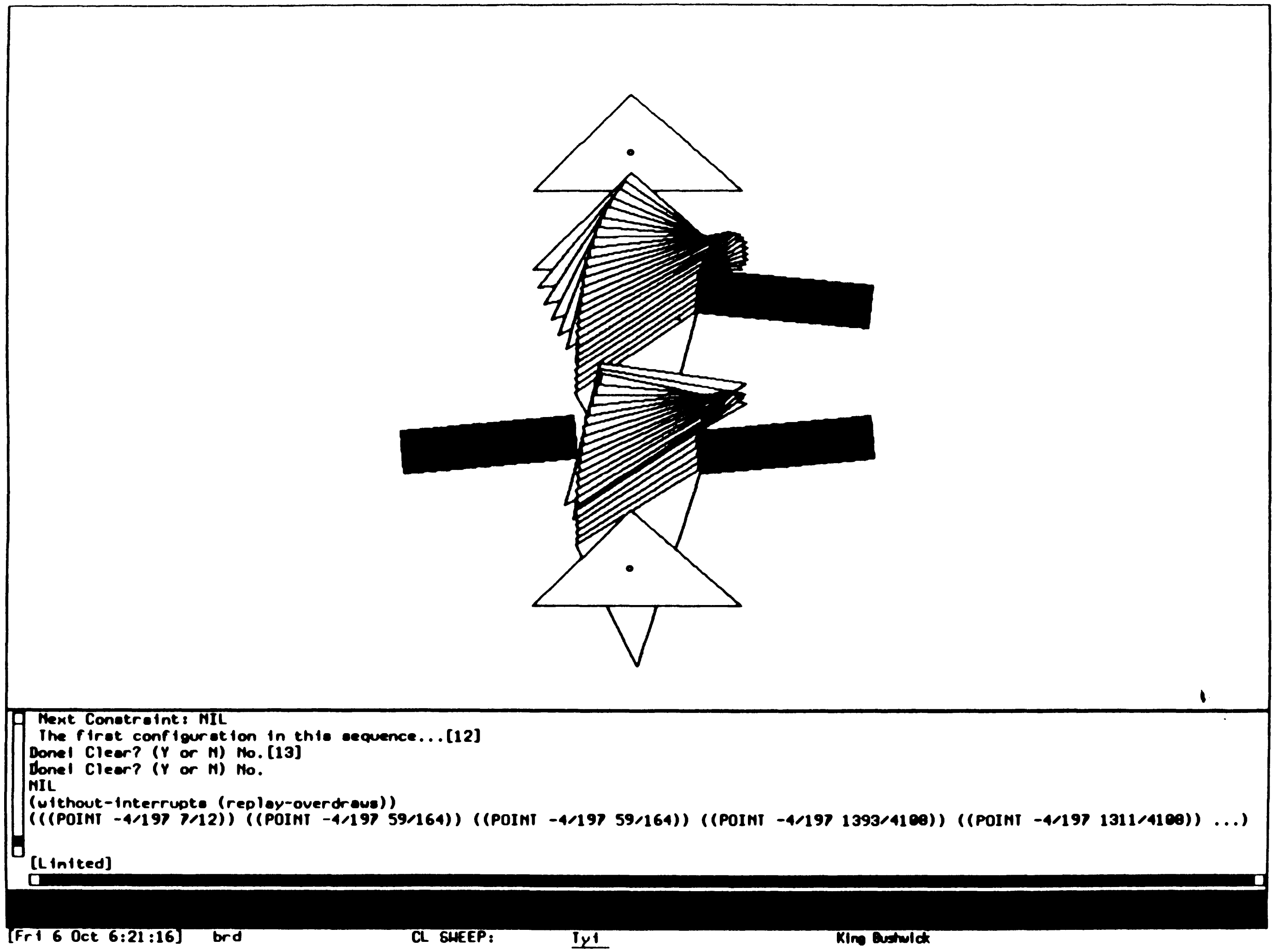

Constraint: (CSURFACE :B 0 0 0 0 -82820/591 -29520/197 -205 205 94 # ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -0065 -4137 4137 -0065 126474/205 -1554921/205 0 0 -2716 # ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :A -0065 -4137 4137 -0065 126474/205 -1554921/205 0 0 -2716 # ...)
Next Constraint: NIL
Next Constraint: NIL
[Linited]
[Fri 25 Aug 2:31:19] brd
CL SWEEP:
Run
King Bushwick's console idle 6 minutes

Figure 4n



Constraint: (CSURFACE :A -8065 -4137 4137 -8065 126474/205 -1554921/205 0 0 -2716 # ...)
Next Constraint: NIL
Next Constraint: NIL
Ok.
Constraint: NIL
Constraint: NIL
Next Constraint: NIL
Next Constraint: NIL
[Limited]
[Fri 25 Aug 2:31:52] brd
CL SWEEP:
Run
King Bushwick's console idle 7 minutes

Figure 4-0



Next Constraint: NIL
The first configuration in this sequence...[12]
Done! Clear? (Y or N) No.[13]
Done! Clear? (Y or N) No.
NIL
(without-interrupts (replay-overdraws))
(((POINT -4/197 7/12)) ((POINT -4/197 59/164)) ((POINT -4/197 59/164)) ((POINT -4/197 1393/4100)) ((POINT -4/197 1311/4100)) ...)
[Limited]
[Fri 6 Oct 6:21:16] brd CL SWEEP: Tyi King Bushwick

Figure 5a

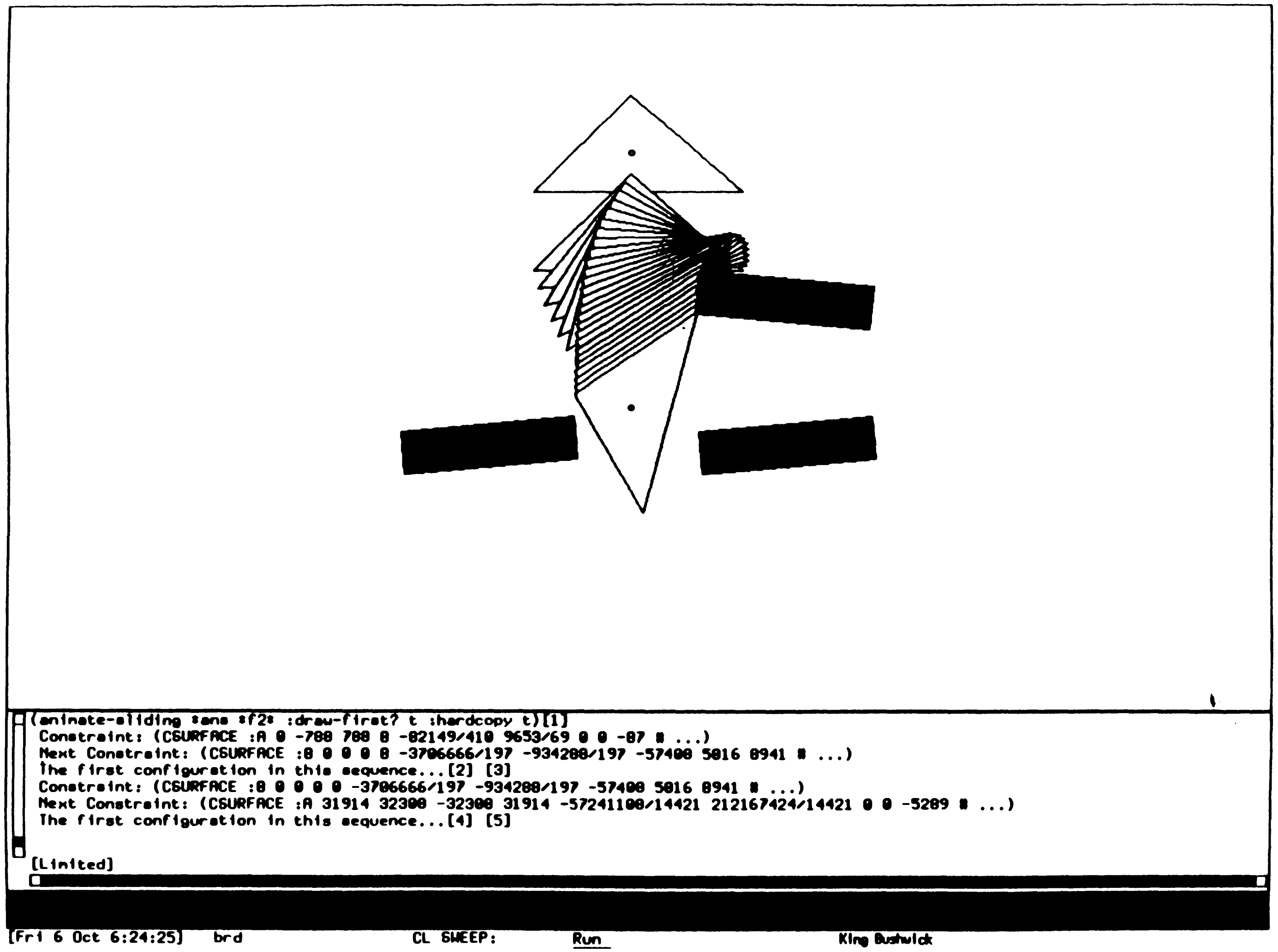
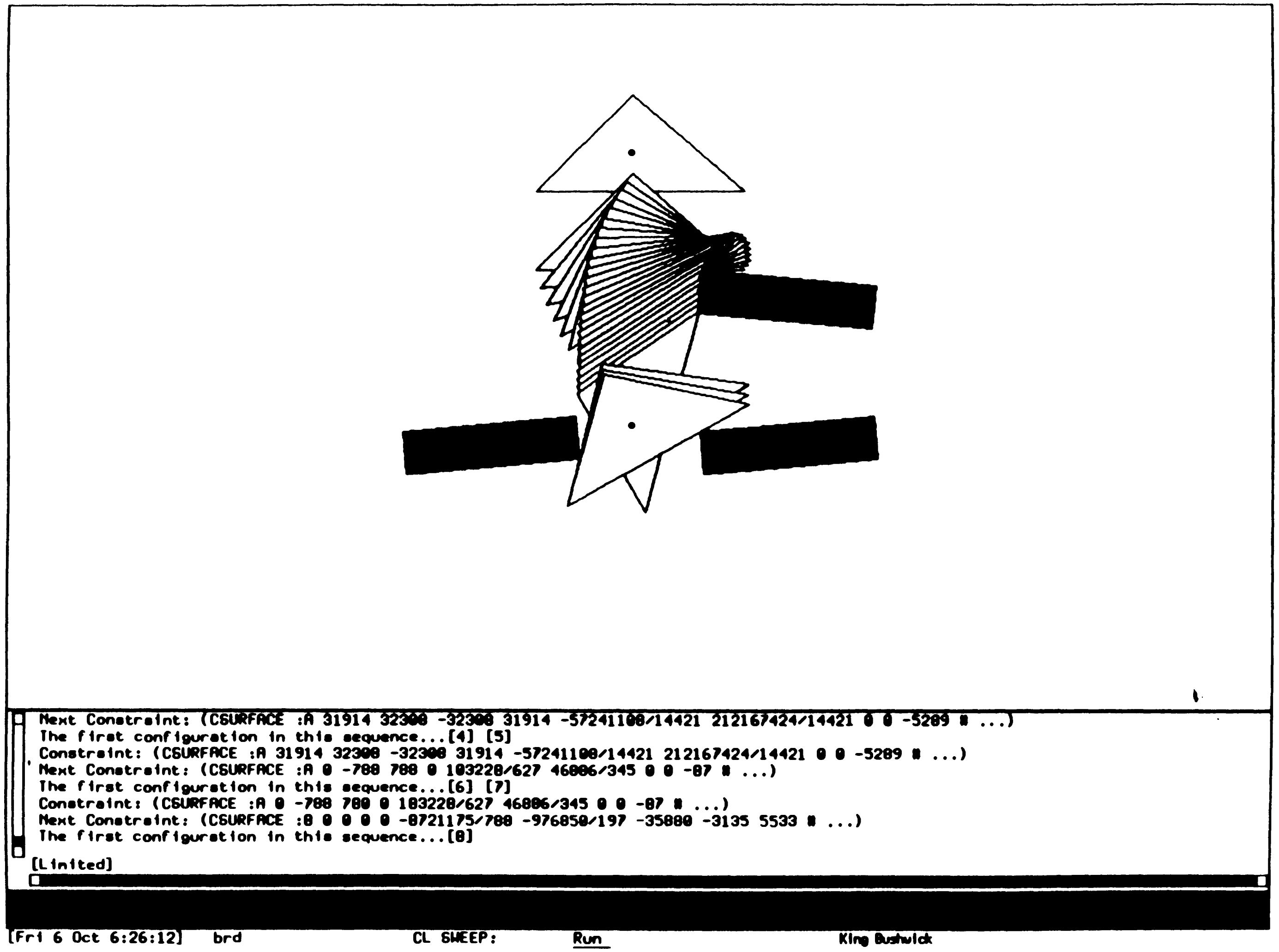
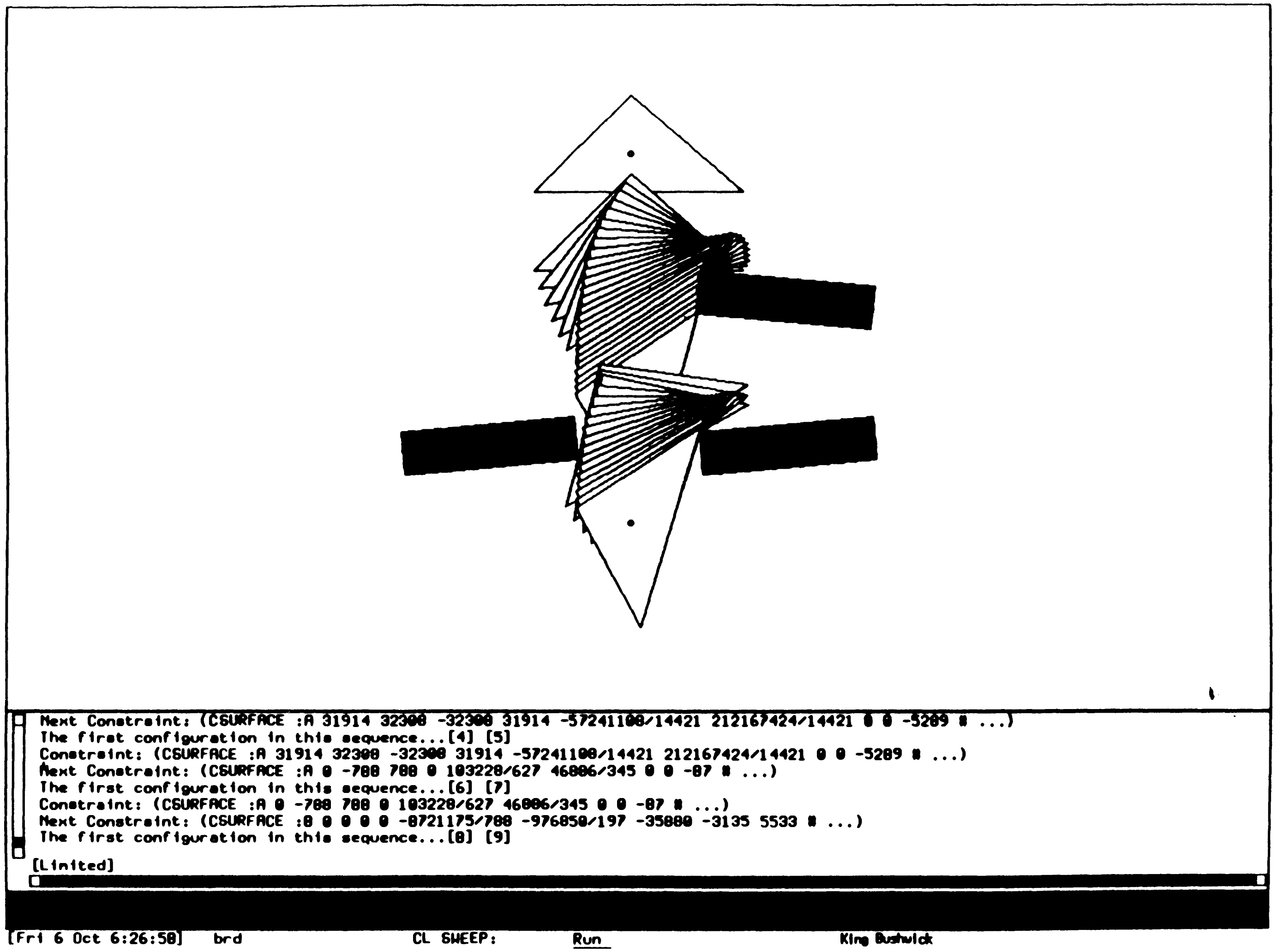

(animate-sliding *ans *f2* :draw-first? t :hardcopy t)[1]
Constraint: (CSURFACE :A 0 -700 700 0 -82149/410 9653/69 0 0 -87 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -3706666/197 -934200/197 -57400 5016 8941 # ...)
The first configuration in this sequence...[2] [3]
Constraint: (CSURFACE :B 0 0 0 0 -3706666/197 -934200/197 -57400 5016 8941 # ...)
Next Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5289 # ...)
The first configuration in this sequence...[4] [5]
[Limited]
[Fri 6 Oct 6:24:25] brd
CL SWEEP:
Run
King Bushwick

Figure 5b



Next Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5209 # ...)
The first configuration in this sequence...[4] [5]
Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5209 # ...)
Next Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46806/345 0 0 -87 # ...)
The first configuration in this sequence...[6] [7]
Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46806/345 0 0 -87 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -8721175/700 -976850/197 -35800 -3135 5533 # ...)
The first configuration in this sequence...[8]
[Limited]
[Fri 6 Oct 6:26:12] brd
CL SWEEP:
Run
King Bushwick

Figure 5c



Next Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5289 # ...)
The first configuration in this sequence...[4] [5]
Constraint: (CSURFACE :A 31914 32300 -32300 31914 -57241100/14421 212167424/14421 0 0 -5289 # ...)
Next Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46006/345 0 0 -87 # ...)
The first configuration in this sequence...[6] [7]
Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46006/345 0 0 -87 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -8721175/700 -976850/197 -35000 -3135 5533 # ...)
The first configuration in this sequence...[8] [9]
[Linited]
[Fri 6 Oct 6:26:50] brd
CL SWEEP:
Run
King Bushwick

Figure 5d

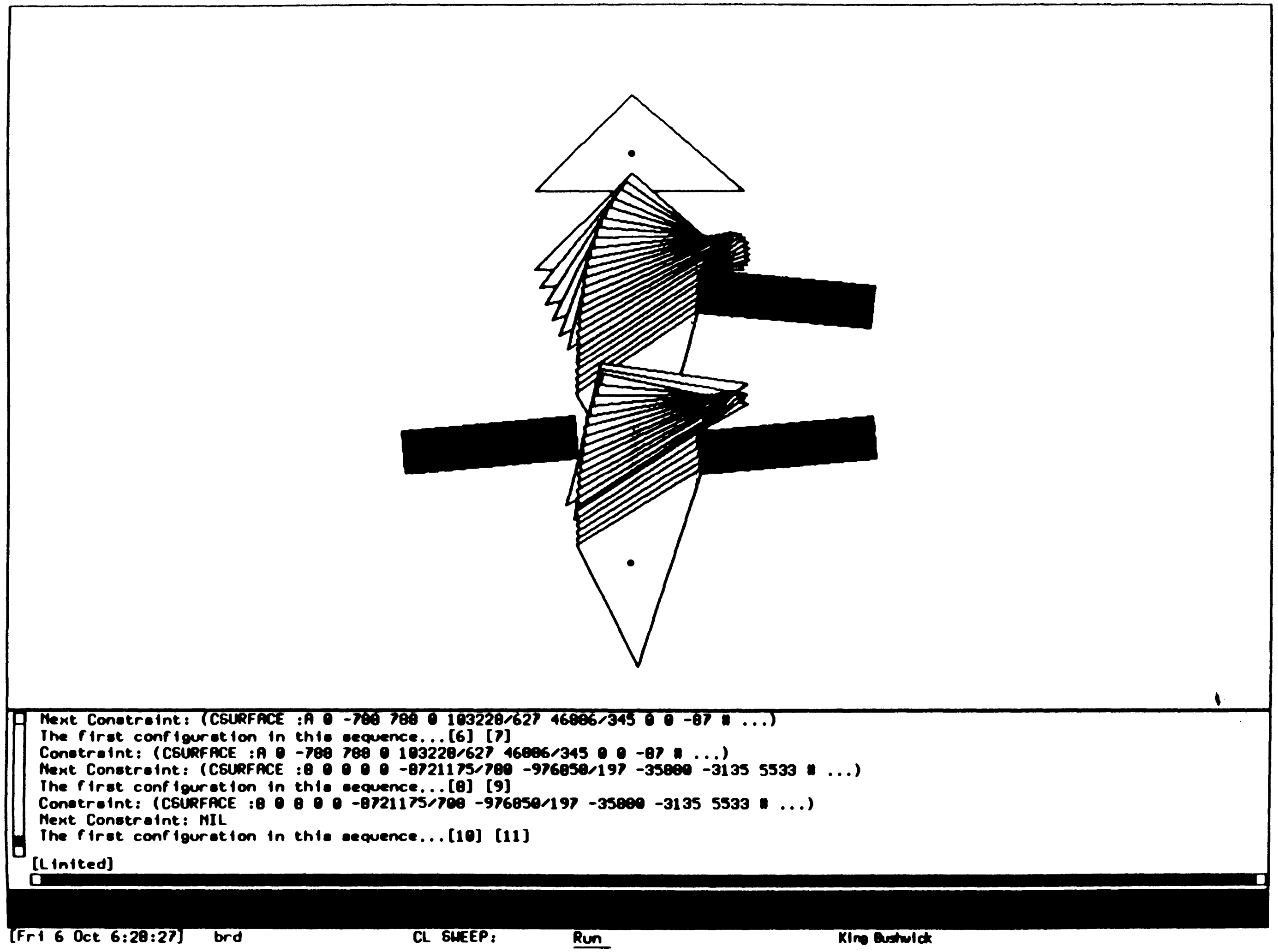

Next Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46806/345 0 0 -87 # ...)
The first configuration in this sequence...[6] [7]
Constraint: (CSURFACE :A 0 -700 700 0 103220/627 46806/345 0 0 -87 # ...)
Next Constraint: (CSURFACE :B 0 0 0 0 -8721175/700 -976850/197 -35800 -3135 5533 # ...)
The first configuration in this sequence...[8] [9]
Constraint: (CSURFACE :B 0 0 0 0 -8721175/700 -976850/197 -35800 -3135 5533 # ...)
Next Constraint: NIL
The first configuration in this sequence...[10] [11]
[Limited]
[Fri 6 Oct 6:28:27] brd
CL SWEEP:
Run
King Bushwick

Figure 5e

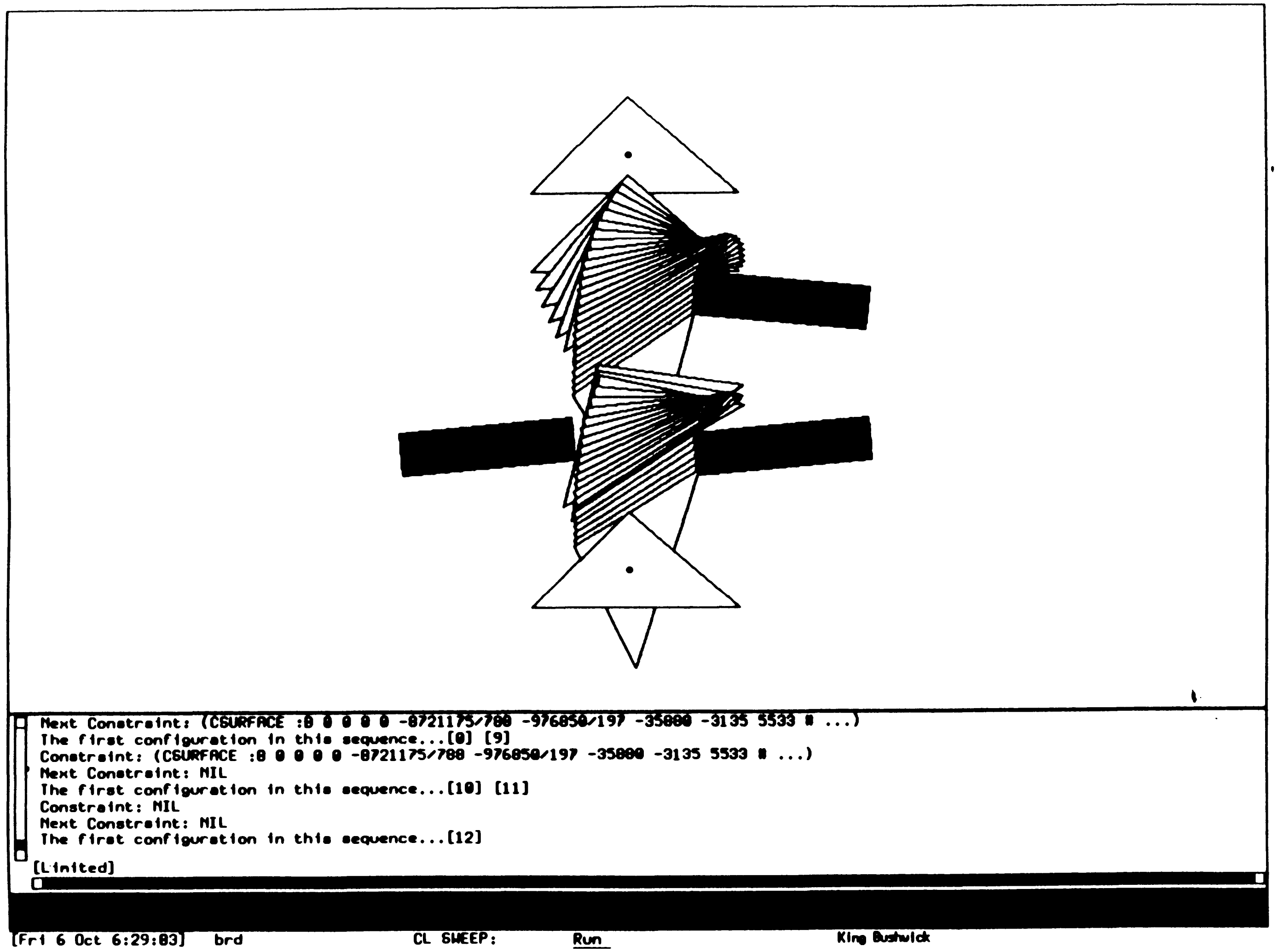

Next Constraint: (CSURFACE :0 0 0 0 0 -8721175/788 -976850/197 -35000 -3135 5533 # ...)
The first configuration in this sequence...[0] [9]
Constraint: (CSURFACE :0 0 0 0 0 -8721175/788 -976850/197 -35000 -3135 5533 # ...)
Next Constraint: NIL
The first configuration in this sequence...[10] [11]
Constraint: NIL
Next Constraint: NIL
The first configuration in this sequence...[12]
[Linited]
[Fri 6 Oct 6:29:03] brd CL SWEEP: Run King Bushwick

Figure 5f

Figure 6: "Motion diode" example. When we try to pull the flexible object back up (move in direction $+y$), it gets stuck due to kinematic constraints.

sliding along A's upper left corner. Eventually, the pawl breaks contact with A, and snaps off, only to hit the rightmost vertex of C. It briefly slides (while rotating compliantly) along C, until it hits B. The tighter constraint from B takes over, and the pawl is again "cocked" counterclockwise until it breaks contact at the lower left vertex of B. Finally, the pawl snaps off B to its rest position.

Now, this mechanical system is a "diode" with respect to pure y-translation (see the figures)—when the pawl is moved back up in the $+y$ direction, it jams due to incompatible kinematic constraints. More interestingly, if B and C are extended to the right and left (resp.), the system is a diode with respect to *all* translational motions. That is, no commanded translation can bring the flexible body back out of the hole between B and C. Our algorithm can decide that for a particular motion plan, a system is a diode. There also exists a theoretical extension of our algorithm, using the theory of real closed fields, which can decide whether the system is a diode with respect to *every* disassembly plan, but this algorithm is not practical. In practice, applying the algorithm to a discretization of the control set would be more practical, although not exact.

4 Overview of Results

4.1 Differential Theories of Mechanics: An Algebraic Approach

We view the motion prediction problem, even with rotational compliance and quasi-static mechanics, as a problem that can be solved by careful reduction to intersection (or collision detection) problems [Can86, Don84,87]. Our approach to modeling rotational compliance and to incorporating frictional constraints leads to the first formulation of the motion prediction problem which permits a *reduction* of motion prediction to collision detection. Our solution differs from previous work on predicting, bounding, and planning rotationally compliant motions with quasi-static mechanics in that it is (i) *purely algebraic*, and hence exact, (ii) *combinatorially precise*, in that the computational complexity is exactly known, (iii) practical and implementable, and (iv) requires no integration. In this subsection we elaborate somewhat on these characteristics.

In order to predict the motion of objects in a mechanical system, we require a computational theory of mechanics. A *differential* theory of mechanics takes the instantaneous state and forces, and computes the resultant instantaneous motion.

Quasi-static analysis is another differential theory of mechanics, in that it predicts the instantaneous motion given state and friction forces.[5] In general, differential theories can be integrated, in the sense that the long-term behavior can be integrated out given the differential mechanics. We say that a theory is *closed-form integrable* when we can solve for the resultant curves in state-space in closed form. When a theory is closed-form integrable,

[5] In the case of ambiguity or non-determinism, we view such a theory as a relation.

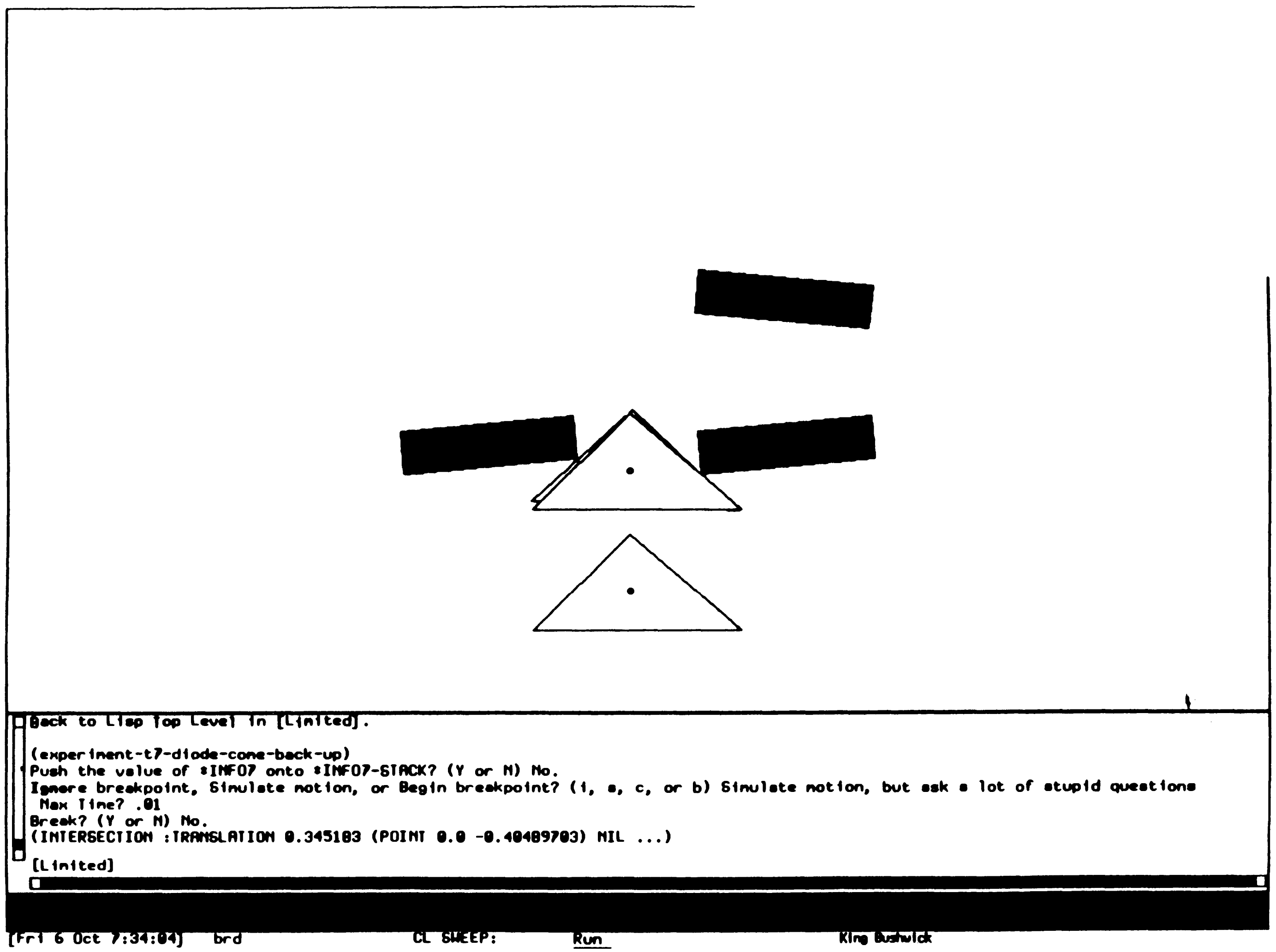

Back to Lisp Top Level in [Limited].
(experiment-t7-diode-cone-back-up)
Push the value of *INFO7 onto *INFO7-STACK? (Y or N) No.
Ignore breakpoint, Simulate motion, or Begin breakpoint? (i, s, c, or b) Simulate motion, but ask a lot of stupid questions
Max Time? .01
Break? (Y or N) No.
(INTERSECTION :TRANSLATION 0.345183 (POINT 0.0 -0.40489703) NIL ...)
[Limited]
[Fri 6 Oct 7:34:04] brd CL SWEEP: Run King Bushwick

Figure 6a

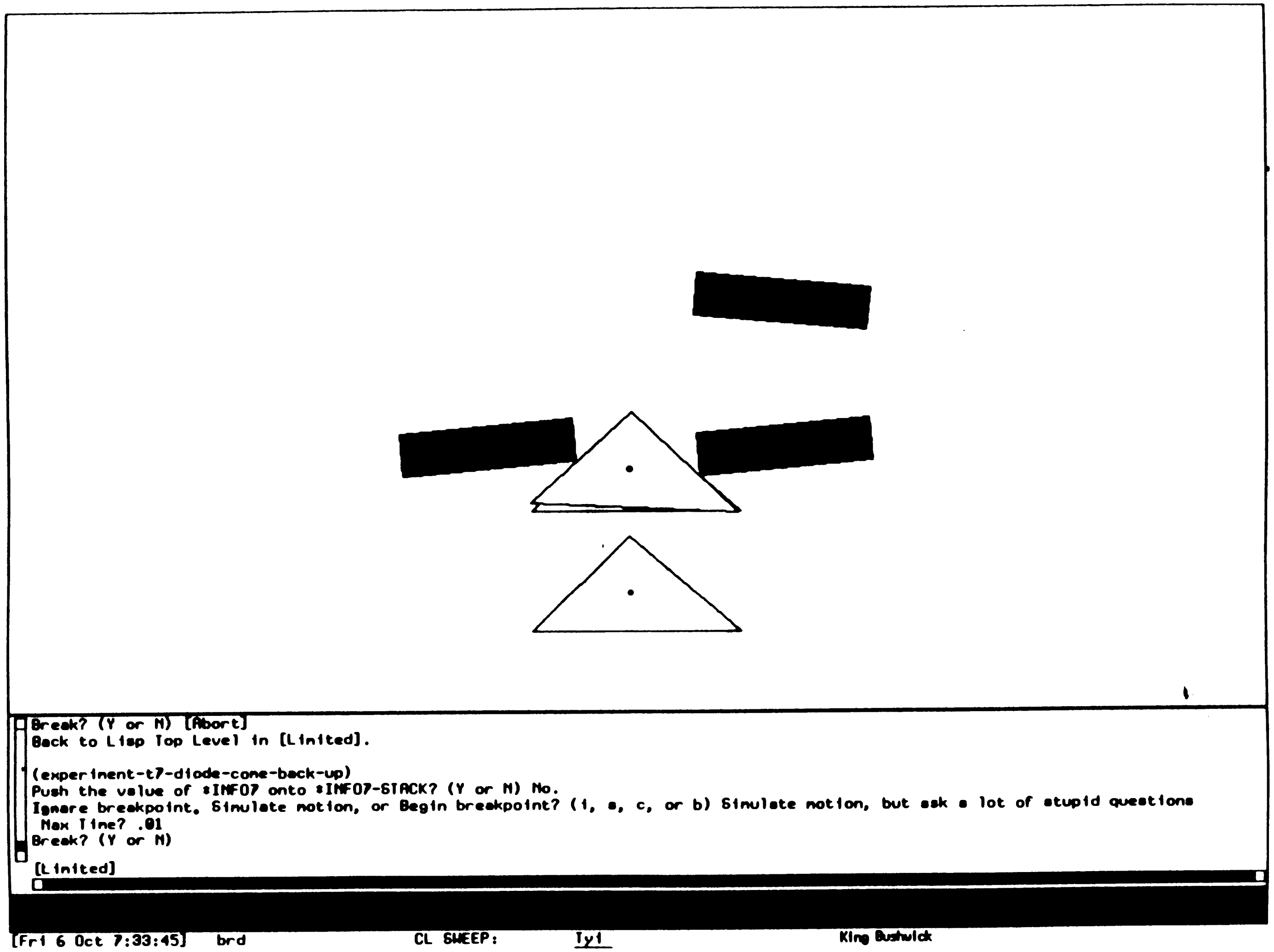

Break? (Y or N) [Abort]
Back to Lisp Top Level in [Limited].
(experiment-t7-diode-cone-back-up)
Push the value of *INFO7 onto *INFO7-STACK? (Y or N) No.
Ignore breakpoint, Simulate motion, or Begin breakpoint? (i, s, c, or b) Simulate motion, but ask a lot of stupid questions
Max Time? .01
Break? (Y or N)
[Limited]
[Fri 6 Oct 7:33:45] brd CL SWEEP: Tyi King Bushwick

Figure 6b

we can develop exact, combinatorially precise algorithms for predicting and planning the motion of objects [Don88a,b, Can89, Briggs89]. When a closed-form solution is not known, numerical methods can be used to integrate out solutions in some cases.

Quasi-static mechanics and generalized damper dynamics are indeed closed form theories in the case of pure translation. However once rotational compliance—the tendency of a manipulated object to change its orientation in response to reaction forces from the environment—-is introduced, we do not have closed-form integrable theories of mechanics. We do have precise, algorithmic theories of the differential mechanics, such as Erdmann's generalized friction cone [Erd84,Erd91] and the acceleration center force-dual model of Brost and Mason [BM89], but these theories must be numerically integrated to obtain solutions. The best algorithms for this process are not combinatorially precise, and do not have solution accuracy bounds. This problem is exacerbated by uncertainty in sensing and control; in this case we are left with the very hard problem of trying to integrate out a stochastic vector field on a holonomic constraint, subject to a family of initial conditions.

In contrast, we use a well-known theory of mechanics with rotational compliance that is not only closed form integrable, but closed form *algebraic*. By this we mean that the following. Our theory is based on the differential mechanics, but can be integrated to produce solutions that are algebraic curves in the configuration space. In fact, the solutions paths are parameterized by time and are piecewise quadratic or linear. Since we can represent configuration space C-surfaces as quadric surfaces, all of the resulting intersection problems (Section 4.2) are no harder than intersecting a quadratic path with a quadric surface. Somewhat surprisingly, even the functions that determine at what times sliding and sticking will occur on a surface are also quadratic in the time parameter.

Closed form, algebraic solutions permit us to construct exact, combinatorially precise algorithms for the motion prediction problem. Furthermore, they allow us to straightforwardly generalize our techniques to encompass certain simple types of uncertainty in control and initial conditions.

In principle, our algorithms can be implemented using exact-precision, algebraic numbers. In practice, we use finite-precision approximation techniques. Robustness is a key issue, and algorithms that are theoretically correct with exact precision are often numerically unstable. A major component of our research consists of building a system that can strengthen the theoretical algorithms (eg, by adding consistency checks) to make them practical.

4.2 Computing Motions and Intersection Problems

We now provide a somewhat informal view of our results. Here are the qualitative states of the root body: it is either undergoing a pure translation, or else it it is stuck due to incompatible kinematic constraints. It is clear that for the root body, all we need to compute is the time at which this sticking occurs. This time is an upper bound on the simulation.

The state of a pawl is more complicated. When a pawl is in free-space undergoing a pure translation, it can strike a surface. This requires a translational collision detection algorithm. When the pawl is in contact with a surface, as time increases, its configuration (orientation in this case) must change so as to comply with this constraint. (We use the term "constraint" as in [LoP, Don87], to refer to the kinematic constraint that (a) an edge of the pawl touch a vertex of the environment or (b) a vertex of the pawl touch an edge of the environment. These constraints form type "(A)" and "(B)" configuration space surfaces, respectively). See, for example, fig. 4. In this case, the configuration space of the flexible object is $\Re^2 \times S^1 \times S^1$; a point in $\Re^2$ specifies the configuration of the root body, whereas each angle in S^1 specifies the orientation of the right and left pawls. In this case, we require an algorithm that can return the mapping from time to configuration, and the curve in configuration space which is the image of this mapping.

Now, as the pawl traces out this curve, three things can happen. First, the pawl may break contact with the surface constraint, due to incompatible kinematics. Second, the pawl may stick on the surface, due to force-balance from the reaction forces. (More generally, for every constraint, there may be at most two disjoint time intervals during which sliding (resp. sticking) occurs, separated by an interval during with sticking (resp sliding) occurs). Third, the pawl may strike another constraint. In Sections 6, 7, and 8, we perform the kinematic and physical analysis required to compute the times at which these events occur, and we describe the algorithms. In particular, we show how, given a constraint, to compute a quadratic function of time whose zeros define the endpoints of these intervals of sliding and sticking behavior.

Finally, when a pawl breaks contact and "snaps off", we must perform a pure rotational intersection test to determine where it snaps to. When a pawl lies on the intersection of two constraints, we must determine whether sticking occurs there due to incompatible constraints, or which constraint takes precedence and becomes a new constraint at that time.

4.3 Exact solutions for mechanical simulations

A major impediment to developing simulation systems has been the apparent necessity to integrate out the differential mechanics in order to determine the long-term behavior of the system. This problem is exacerbated by the fact that in many models of rotational compliance such as the generalized damper (eg., [LMT, Erd, Don2, Don3, Can2]), the resulting trajectories are not known to be algebraic; neither do we have ways of computing algebraic bounding approximations (or forward projections). Hence the traditional numerical approach to simulating such systems has been the following:

Typical Simulation Algorithm

1. Given a state x of the system, numerically integrate the differential equation governing motion of the system. Step forward in time to obtain (approximately) new state x'.
2. Perform collision detection either at x' or along the path from x to x'.
3. If the constraints have changed, reformulate the differential equation.
4. Repeat.

Numerical simulation of mechanical systems is fraught with error, special cases, and numerical problems. They are rarely combinatorially precise, and almost never come with guarantees of accuracy. In this paper, we show how our simple model of compliance permits us to obtain algebraic, closed-form solutions to simulation problems for a rotationally compliant object, and how this leads to *exact* algorithms for analyzing designs for assembly. In particular, the linear map $\boldsymbol{p}(t)$ is given by eq. (1), and once we "rationalize" rotations via the standard substitution $u = \tan\frac{\theta}{2}$, each map ϕ_h is piecewise-cubic. Thus numerical simulation can, in principle, be avoided, and exact solutions can be obtained. We cannot claim that this can be done in general. However, our method yields, in this case, computationally efficient, exact solutions, and may possibly be useful in other domains.

In Section 9, we give a new simulation algorithm that is exact, and runs in time $O(k\lambda_r(n)\log^2 n)$. Here, we assume that the obstacles have m_b vertices and the moving object (root and one pawl) has m_a vertices; we define $n = m_a m_b$ to be the measure of the geometric complexity. $\lambda_r(n)$ is the (almost linear) maximum length of (n, r) Davenport Schinzel sequences [GSS] and r is a small constant. Our method reduces the simulation problem to a plane sweep of an arrangement of algebraic curves in configuration space.

4.4 A Classification of Motion Diodes

Consider figure 7, which is a gedankenexperiment illustrating different types of diodes. The flexible mechanism, and in particular the pawl structure is very similar to fig. 3. The commanded motion is in the $-y$ direction. Now, for geometry (b), the pawl deflects clockwise, and snaps onto the sharp bump at the lower left of (b). The motion is not reversible, due to kinematic constraints alone. Now, unless constraint (c) is present, there may exist other translations which will disassemble the mechanism. The addition of constraint (c) means that no translation can disassemble the mechanism (so long as (b) is "tall enough"—see fig. 7). Now, consider 7d. This geometry also prevents $+y$ motion, but the sticking is due to friction, not kinematics. Finally, consider 7e. Here, relatively small forces are required on the root in the $-y$ direction to assemble the object; however, large forces are required on the root in the $+y$ direction to disassemble the object. (e) is an object that it is easier to assemble than it is to disassemble.

One application of our algorithm has been to determine whether a system is a motion diode, and what kind of diode it is. Flexible mechanisms which function as motion diodes can be used to fasten one part to another, and to make the mechanical connection robust with respect to attempted relative motion of the two parts. This kind of analysis could be very useful in design for assembly. Using our algorithm, we can form the following *classification* of flexible mechanism motion diodes. The classification is "two dimensional" in the sense that one "axis" is relative vs. total motion, and the other axis is Kinematic vs friction vs force diodes.

More specifically, the first classification is

1. A *relative motion diode* is a (flexible object, environment) pair such that for some family of controls, called the *relative motions,* no change in the sign of the controls can reverse the motion to reachieve the start position.

2. A *total motion diode* is a relative motion diode for all relative motions.

The second classification is:

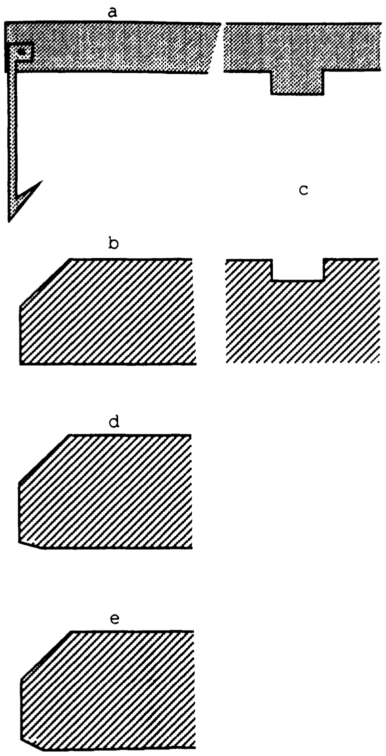


Figure 7: In (a), we see a pawl-type mechanism affixed to a root body which is translating down in the $-y$ direction. The pawl is very similar to fig. 3. (b) forms a relative motion kinematic diode. Without constraint (c), there exist other translations that disassemble the "snapped onto" final configuration. If we make the distance *top* long enough so that the root collides with the top of b, then the addition of constraint (c) turns (b) into a total-motion kinematic diode. Now, (d) forms a friction diode, and (e) forms a force diode. The presence or absence of (c) determines whether (d) and (e) are total or relative motion diodes.

A A *kinematic* diode prevents reverse motion due to purely kinematic constraints.

B A *friction* diode prevents reverse motion due to a combination of kinematic constraints and coulomb friction. It depends on the coefficient of friction.

C A *force* diode is described in sec. A. Essentially, one imagines replacing the control of a root body by generalized spring position control, thus equating displacements of and forces on the root body in a first-order relation. This permits one to ask *What forces are experienced by the pawls as a function of time?* and *What force is exerted on the root to cause the motion?* A force diode, roughly speaking, is a friction diode for all control forces below a certain modulus bound.

Hence, we see that without friction, figs. 3-4 is not a diode. With sufficiently high friction, it is a relative motion friction diode. Fig. 5 is a total motion kinematic diode. Our algorithm can decide all these classifications automatically.

In figure 7, we see that (b) is a kinematic diode, (d) is a friction diode, and (e) is a force diode. If we add constraint 7c and make (b), (d), and (e) "tall enough", then these are total motion diodes; otherwise they are relative.

One goal of our algorithm has been to provide an algorithmic means of classifying flexible objects such as snap-fasteners by diode type. Obviously, this is just a start, and other, finer types of classifications are possible. Ours seems useful in design, and is efficiently computable by implemented, precise algorithms.

PART II: DETAILS AND ANALYSIS

5 Simulation and Algebraic Intersection Problems

From an algorithmic point of view, our work can be thought of as reducing the flexible object motion prediction problem to intersection problems (like collision detection) in Cspace. There are, of course, additional complications such as the presence of friction which need to be addressed. In this section, we show how to reduce geometric intersection problems to problems in elementary elimination theory (particularly, the simultaneous solution of polynomial equations and inequalities).

As the linked body $\mathcal{M}$ is moved in the environment $\mathcal{N}$ (see Figure 2), the pawl $\mathcal{M}_h$ may collide with an obstacle. We need robust algorithms to detect and report the collision. While a pawl is sliding on a feature of the environment, it may arrive at configuration at which it is kinematically impossible to continue to maintain contact – the pawl will either snap off the environment or jam on it. We need a way to determine when this occurs. These seemingly unrelated problems are in fact reducible to intersection problems in configuration space.

Three types of motion are possible for the pawl in our problem: pure translation without rotation; pure rotation without translation; and most importantly, the motion of the pawl sliding while maintaining contact with a given feature of the obstacle. The collision detection problem is to find the times at which the pawl collides with $\mathcal{N}$, and report which features are in contact during collision.

We are able to perform this analysis efficiently by developing a simple dynamical systems model using local dynamic constraints. These tools permit us to reduce the simulation to a plane sweep of a "dynamically annotated" slice of configuration space.

More specifically, two types of contact are possible between the pawl $\mathcal{M}_h$ and a polygon in $\mathcal{N}$. Following the convention of Lozano-Pérez [LP83] we say that Type-A contact occurs when a vertex of $\mathcal{N}$ touches an edge of the pawl; Type-B contact occurs when a vertex of the pawl touches an edge of $\mathcal{N}$.

We can now write the contact constraint equations for the two types of contact, as in [Can86]. We shall index features (vertices and edges) of the moving pawl by the subscript i and features of the polygonal environment by the subscript j. Let an edge of $\mathcal{M}_h$ be represented by its outward normal, $\boldsymbol{n}_i$, and its distance to the hinge point along the normal, d_i. Let $\boldsymbol{p}_j$ be a vector to the contact vertex of $\mathcal{N}$. Let R_θ be the linear transformation which rotates a vector by an angle θ. Then the type-A constraint can be written as

$$(\boldsymbol{p}_j - \boldsymbol{p}) \cdot R_\theta \boldsymbol{n}_i - d_i = 0. \tag{3}$$

Similarly, the type-B constraint can be written as

$$(R_\theta \boldsymbol{p}_i + \boldsymbol{p}) \cdot \boldsymbol{n}_j - d_j = 0. \tag{4}$$

Where $\boldsymbol{p}_i$ is the vector from the hinge point P_h to the contact vertex of $\mathcal{M}_h$.

In each of the cases, the obstacle constraint equations (3 and 4) and can be reduced to algebraic equations. This is possible since the trigonometric functions in the rotation matrix R_θ can be replaced by new indeterminate quantities (eg. $\sin\theta = s$, $\cos\theta = c$), with the addition of a new constraint ($s^2 + c^2 - 1 = 0$). More practically, the substitution $u = \tan\frac{\theta}{2}$ yields constraint equations that are quadratic in u, with coefficients that are affine in x and y. Since x and y are affinely parametric in time t, the coefficients are also affine in t. Intersecting two of these constraints requires intersecting two quadratics. Pure rotational intersection detection (the "snap") requires solving a quadratic in u. Pure translational intersections require solving an affine equation. Hence there exists a closed-form, purely algebraic solution to these intersection problems. The collision detection for each obstacle constraint can be done in constant time, since the degree, size, and number of variables in the constraint polynomials is fixed. If m is the geometric complexity of the pawl and n is that of the size of the environment, there are mn such constraints.

Since the constraint equations are quadratic in u, it possible that in some interval of time the discriminant Δ of the constraint equation (treated as an quadratic in u) becomes negative, yielding no solution for u. This corresponds to the pawl either having snapped off the constraint or having jammed on it. By solving for the critical times t_z at which the discriminant changes sign, we can determine, by careful analysis, which case occurs. The details of this analysis are found in Section 8.3, since it is related to the problem of choosing the "correct branch" for collision detection.

The geometric intersection algorithms for our problem, however, involve several subtle and important issues to make them robust in practice. We go into some detail about how to solve such systems as specialized to our particular application, and how to implement the solution. We have two goals. The first is to elucidate a theoretical, exact, algebraic algorithm that is correct. However, this algorithm is numerically unstable when implemented with finite precision arithmetic. Hence robustness is a key issue. We emphasize the steps we have taken in order to enhance robustness, and, in particular, to strengthen the theoretical algorithm by adding consistency checks.

Our approach to implementing a robust algorithm is somewhat experimental. The first step is noticing what robustness difficulties arise. The second is analyzing their causes, and placing this analysis on a firm mathematical footing. Where possible, we propose solutions for some of the cases in which we believe we have found a robust strengthening of the underlying theoretical algorithms.

We implemented a specialization of Donald's representation [Don 84, 87] for 6 DOF ($\Re^3 \times SO(3)$) configuration space obstacles to the case of $\Re^2 \times S^1$. We quickly review that representation. Let C denote the configuration space $\Re^2 \times S^1$, and let $(\boldsymbol{x}, \theta)$ be a typical configuration. A Cspace obstacle CO is defined by a predicate on configurations (as in [Don 87])

$$\bigwedge_{i \in cfamily(A,B)} \Big(\theta \in \mathcal{A}_i \Longrightarrow f_i(\boldsymbol{x}, \theta) \leq 0\Big). \tag{5}$$

Here, the functions $f_i : C \to \Re$ are called C-functions; their negative conjunction in effect defines the Cspace obstacle. Each f_i is restricted by an applicability region $\mathcal{A}_i$ which is a sector (angular interval $[\theta_{min}, \theta_{max}]$) of the unit circle S^1. The reason for this is that each f_i is generated by considering the interaction of a feature (edge or vertex) of a pawl polygon A with a feature (vertex or edge) of an obstacle polygon B. Contact, or interaction between generating features is only possible for a connected angular interval of orientations; this interval is precisely the applicability interval $\mathcal{A}_i$. The set of all C-functions generated by A and B is called the "C-family" $cfamily(A, B)$. See [BLP, Don87] for details on computing the C-functions and the applicability constraints.

A C-surface $\ker f_i$ is defined as the applicable zero-set (kernel) of a C-function f_i. It contains a patch of configurations where the two generating features of f_i can be placed in contact. The key step in detecting a collision of a path with a cspace obstacle CO defined by (5) is to simultaneously solve for the path's parameter subject to the csurface constraint[6] $f_i = 0$.

6 Sticking due to Friction

A careful analysis of the physics of compliant sliding is necessary to formulate a precise, combinatorial version of this apparently continuous problem. When the pawl is in contact with the environment, it is possible for the motion to stop because the contact forces are adequate to balance the applied forces. This is called "sticking due to friction," to distinguish it from other kinds of sticking. The algorithm must determine if this can happen during motion. The work of Mason [Mas82] and Erdmann [Erd84] addresses this issue in considerable detail. In this section, we show that for our problem this determination can be made by a simple geometric test.

Coulomb's Law of friction states that the tangential force due to friction, $\boldsymbol{f}_t$, on a point which is sliding on an edge in a specified direction is given by $\boldsymbol{f}_t = \mu \boldsymbol{f}_n$, where $\boldsymbol{f}_n$ is the force normal to the edge and μ is the coefficient of friction. In addition, the direction of the friction force is opposite that of the motion.

Coulomb's Law has a geometric interpretation which is frequently more useful (see, for example, [Mas82]). We provide here a slightly different interpretation, which is more applicable to our problem. Suppose point $\boldsymbol{p}_i$ is in contact with edge e_j. The outward pointing unit normal vector to the edge is denoted $\boldsymbol{n}_j$. We are also given the direction of sliding by specifying $\boldsymbol{v}$, the unit vector tangent to the edge in the direction of sliding (see Figure 8). Coulomb's Law defines a half-cone of forces acting on the point $\boldsymbol{p}_i$. The cone is the convex combination [7] of the vectors $\boldsymbol{n}_j$ and the vector $\boldsymbol{f}_e = \boldsymbol{n}_j - \mu \boldsymbol{v}$. Any force in the interior of this cone and along $\boldsymbol{n}_j$ can be resisted and the point will stick. A force along $\boldsymbol{f}_e$ will result in $\boldsymbol{p}_i$ sliding in equilibrium, in the direction of $\boldsymbol{v}$. A force outside the cone results in the point sliding along $\boldsymbol{v}$, out of equilibrium, which is precluded by our assumption of quasi-static motion.

[6] We will often use the term "constraint" to blur the distinction between a C-function and its kernel.

[7] The convex combination of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is the set of all vectors of the form $\zeta \boldsymbol{x} + \eta \boldsymbol{y}$, where $\zeta, \eta \geq 0$.

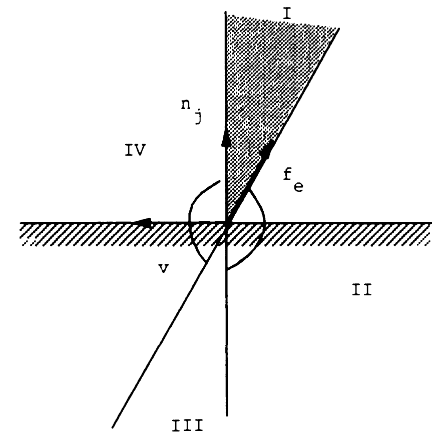


Figure 8: Geometric Interpretation of Coulomb's Law

We first consider the type-B contact: the point is the vertex i of the pawl, and the edge is the edge e_j of the environment. Let $\boldsymbol{r} = R_\theta \boldsymbol{p}_i$. The assumption that stable contact is maintained (see Section 2) between the point and the edge restricts the possible values of the force $\boldsymbol{f}$ on the pawl and the torque $\boldsymbol{r} \times \boldsymbol{f}$ [8] due to the contact force. First, $\boldsymbol{f} \cdot \boldsymbol{n}_j$ must be positive; second, if the hinge is in the right half-plane, we require $\boldsymbol{r} \times \boldsymbol{f} < 0$ for stability, and if it is in the left half-plane, we require that $\boldsymbol{r} \times \boldsymbol{f} > 0$. For quasi-static sliding, the total force is given by $\boldsymbol{f} = \alpha \boldsymbol{f}_e$, $\alpha > 0$. Therefore $\boldsymbol{f}$ trivially satisfies the condition $\boldsymbol{f} \cdot \boldsymbol{n}_j > 0$. The condition on the torque $\boldsymbol{r} \times \boldsymbol{f}$, however, depends on the location of the hinge point. Let the possible locations of the hinge point with respect to the contact point, $-\boldsymbol{r}$, be divided into four sectors as follows: if $-\boldsymbol{r} = \beta \boldsymbol{n}_j + \gamma \boldsymbol{f}_e$, the hinge point is in Sector

I if $\beta > 0, \gamma > 0$,
II if $\beta < 0, \gamma > 0$,
III if $\beta < 0, \gamma < 0$,
IV if $\beta > 0, \gamma < 0$.

Figure 8 also indicates the sectors.

Proposition 6.1 *For type-B contact, the contact point will slide when the hinge is in sectors II and IV. The contact point will not slide (will stick due to friction) when the hinge is in sectors I and III.*

Proof. Suppose the pawl is sliding quasi-statically. Then,

$$\begin{aligned} \boldsymbol{f} &= \alpha \boldsymbol{f}_e \\ &= \alpha(\boldsymbol{n}_j - \mu \boldsymbol{v}). \\ \boldsymbol{r} &= -\left[\beta \boldsymbol{n}_j + \gamma \boldsymbol{f}_e\right] \\ &= -\left[(\beta + \gamma)\boldsymbol{n}_j - \gamma \mu \boldsymbol{v}\right]. \end{aligned}$$

[8] Here $\boldsymbol{r} \times \boldsymbol{f}$ is the component of the usual cross product along the axis orthogonal to the plane, i.e., in coordinates, $\boldsymbol{r} \times \boldsymbol{f} = r_x f_y - r_y f_x$.

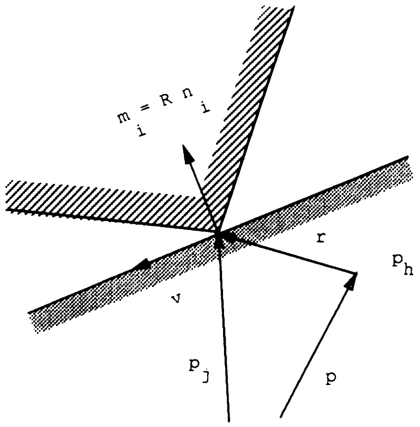


Figure 9: Type-A Contact

$$\begin{aligned} \boldsymbol{r} \times \boldsymbol{f} &= -\alpha\left[0-(\beta+\gamma)\mu+\gamma\mu+0\right] \\ &= \alpha\beta\mu. \end{aligned} \tag{6}$$

Since $\alpha, \mu > 0$, the sign of $\boldsymbol{r} \times \boldsymbol{f}$ is just that of β. Now, in sectors I and II, the right-half plane, we require that $\boldsymbol{r} \times \boldsymbol{f} < 0$ to maintain stable contact. But from Equation 6, $\boldsymbol{r} \times \boldsymbol{f} > 0$ in sector I and $\boldsymbol{r} \times \boldsymbol{f} < 0$ in sector II. Hence the contact point will stick in sector I and slide in sector II. A similar argument shows that the contact point will stick in III and slide in IV. □

A similar argument applies to type-A contact if the sectors and related terms are defined appropriately (see Figure 9). Let $\boldsymbol{f}$ be, as before, the force on the point of contact, which is now a vertex j of the environment, and let $\boldsymbol{r}$ be the vector from the hinge point to the contact point, which is now given by $\boldsymbol{r} = \boldsymbol{p}_j - \boldsymbol{p}$. Let the outer normal of the edge e_i be denoted by $\boldsymbol{m}_i = R_\theta \boldsymbol{n}_i$, $\boldsymbol{v}$ be the unit vector tangent to the edge in the direction in which the vertex slides with respect to the pawl's edge and $\boldsymbol{f}_e = \boldsymbol{m}_i - \mu\boldsymbol{v}$. To maintain contact, $\boldsymbol{f} \cdot \boldsymbol{m}_i$ must be positive and, if the hinge is in the right half-plane, we require $\boldsymbol{r} \times -\boldsymbol{f} > 0$, i.e.$\boldsymbol{r} \times \boldsymbol{f} < 0$, and if it is in the left half-plane, we require that $\boldsymbol{r} \times \boldsymbol{f} > 0$.

With the terms defined as above, we can divide the possible locations of the hinge point with respect to the contact geometry into four sectors exactly as we did for Type-B contact, despite the fact that the interpretation of the sectors is different. Now it is easy to prove the corresponding proposition:

Proposition 6.2 *For type-A contact, the contact point will slide when the hinge is in sectors II and IV. The contact point will not slide (will stick due to friction) when the hinge is in sectors I and III.*

Proof. The proof is textually identical to the proof for Type-B contact, with $\boldsymbol{n}_j$ replaced by $\boldsymbol{m}_i$. □

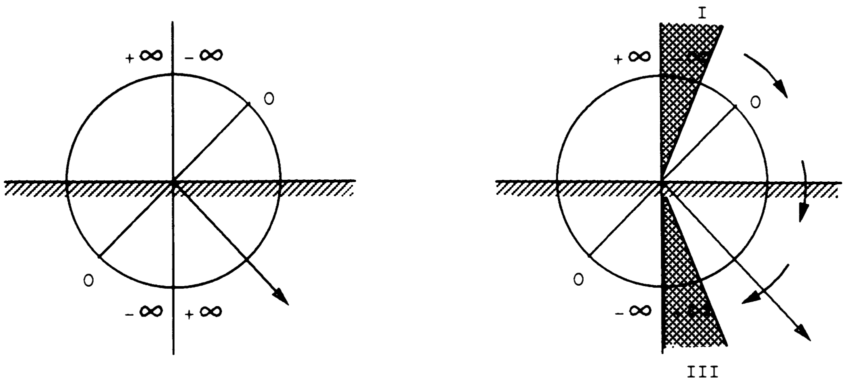


Figure 10: The sliding direction for Type-B contact.

7 The Sliding Direction

The analysis of Section 6 utilized the direction of motion to determine whether the pawl could stick due to friction. It turns out that the direction of the motion can be determined from the differential kinematics of the contact, given the motion of the hinge, $\boldsymbol{p}$.

Consider the motion for a type-B contact. For convenience, assume that the Y-axis of the reference coordinate frame is along the outward normal of the edge, i.e., $\boldsymbol{y} = \boldsymbol{n}_j$, and the X-axis is along the edge, oriented to form a right-handed coordinate system. Recall that $\boldsymbol{r} = R_\theta \boldsymbol{p}_i$. Define the *departure speed*, u, as the component of the velocity of the contact point along the edge normal $\boldsymbol{n}_j$. The velocity of the contact point is given by $\dot{\boldsymbol{r}} + \dot{\boldsymbol{p}}$. Hence

$$u = (\dot{\boldsymbol{r}} + \dot{\boldsymbol{p}})\cdot\boldsymbol{n}_j = \omega r_x + \dot{p_y}. \tag{7}$$

The differential kinematic constraint for sliding corresponds to the requirement that the departure speed is zero.

$$\omega r_x + \dot{p_y} = 0. \tag{8}$$

The *sliding speed*, s, is defined as the component of the velocity of the contact point in the $\boldsymbol{x}$ direction (the vector $\boldsymbol{v}$ of Section 6 is therefore $sgn(s)\boldsymbol{x}$). Hence,

$$\begin{aligned} s &= (\dot{\boldsymbol{r}} + \dot{\boldsymbol{p}})\cdot\boldsymbol{x} & (9)\\ &= -\omega r_y + \dot{p_x}. & (10) \end{aligned}$$

Combining with Equation 8,

$$s = \dot{p_x} + \frac{\dot{p_y}}{r_x} r_y. \tag{11}$$

Equation 11 has a simple geometric interpretation, shown in Figure 10. The circle represents the possible positions of the hinge point with respect to the contact point. The numbers on the circle indicate the sliding speed, when $\dot{p_y} < 0$ (if $\dot{p_y} > 0$, the speeds shown have opposite sign; if $\dot{p_y} = 0$, s is constant). Notice that the sliding speed is 0 when $\boldsymbol{r}\cdot\dot{\boldsymbol{p}} = 0$. Also, from Equation 8, $\omega < 0$ when the hinge point ($-\boldsymbol{r}$) is in the right half-plane of the figure, and $\omega > 0$ in the left half-plane.

We can now combine the above results with those of section 6. Suppose, without loss of generality, that the hinge point is in the right half-plane. If the hinge is in the region corresponding to $s < 0$, the possible direction of slide is to the left. If, in addition, the hinge is in sector I, it will stick. If it is outside sector I, the location of the hinge will move in the direction of ω, at some point (corresponding to $\boldsymbol{r}\cdot\dot{\boldsymbol{p}} = 0$), the direction of slide will reverse to $s > 0$. This also brings sectors III and IV to the right half-plane. Eventually, the hinge will enter sector III for $s > 0$, and stick. Figure 10 also depicts the evolution of the hinge point.

The time at which the contact will stick in sector III is easily computed by letting $R_\theta \boldsymbol{p}_i = \frac{|\boldsymbol{p}_i|}{1+\mu^2}(\boldsymbol{n}_j + \mu \boldsymbol{v})$ in the constraint equation 4 and solving the resulting linear equation for t.

A similar but slightly more complicated analysis applies to Type-A contact. We shall first derive an expression for the velocity $\boldsymbol{w}$ of the contact point with respect to a frame fixed on the moving pawl, and expressed in the world coordinate frame. We shall assume for simplicity of derivation, that the coordinate frame attached to the pawl has its Y-axis along the edge normal $\boldsymbol{m}_i = R_\theta \boldsymbol{n}_i$, and the X-axis is along the edge so as to form a right-handed coordinate system. As in Section 6, for Type-A contact, $\boldsymbol{r} = \boldsymbol{p}_j - \boldsymbol{p}$, and let ${}^p\boldsymbol{r} = {R_\theta}^T \boldsymbol{r}$ be the corresponding vector in the coordinate frame fixed to the pawl. Let Ω be the skew-symmetric angular velocity matrix, given by

$$\Omega = \begin{pmatrix} 0 & -\omega \\ \omega & 0 \end{pmatrix}. \tag{12}$$

where ω is the angular velocity of the pawl. From elementary kinematics (see, for example, [BR79]), $\Omega = \dot{R}_\theta {R_\theta}^T$.

The velocity in the frame attached to the pawl, ${R_\theta}^T \boldsymbol{w}$, can now be calculated by differentiation:

$$\begin{aligned} {R_\theta}^T \boldsymbol{w} &= \dot{{}^p\boldsymbol{r}} \\ &= {R_\theta}^T \dot{\boldsymbol{r}} + \dot{{R_\theta}^T} \boldsymbol{r} \\ &= {R_\theta}^T \dot{\boldsymbol{r}} + {R_\theta}^T \Omega^T \boldsymbol{r}. \\ &\text{Therefore,} \qquad (13) \\ \boldsymbol{w} &= \dot{\boldsymbol{r}} + \Omega^T \boldsymbol{r}. \qquad (14) \end{aligned}$$

The differential constraint corresponds to the departure speed, u, being zero. Therefore,

$$\begin{aligned} u &= \boldsymbol{w}\cdot\boldsymbol{m}_i \\ &= \dot{r}_y - \omega r_x \\ &= 0. \end{aligned} \tag{15}$$

Solving for ω,

$$\begin{aligned} \omega &= \frac{\dot{r}_y}{r_x} \\ &= \frac{-\dot{p}_y}{r_x}. \end{aligned} \tag{16}$$

Hence, the sliding speed, s, is given by

$$\begin{aligned} s &= \boldsymbol{w} \cdot \boldsymbol{x} \\ &= \dot{r_x} + \omega r_y \\ &= \dot{r_x} + \frac{-\dot{p_y}}{r_x} r_y \\ &= -\dot{p_x} - \dot{p_y} \frac{r_y}{r_x} \end{aligned} \tag{17}$$

The map of sliding speeds for Type-A contact is almost identical to Figure 10, except that the direction of the vector $\dot{\boldsymbol{p}}$ is reversed. However, the orbit of the hinge point during sliding is more complicated than in the type-B case. The time at which the hinge point will enter Sector III and stick can be computed as follows. At the time of entry into Sector III, the $\boldsymbol{r}$ is given by

$$\begin{aligned} \boldsymbol{r} &= \frac{|\boldsymbol{r}|}{\sqrt{1+\mu^2}} (\boldsymbol{m}_i - \mu \boldsymbol{v}) \\ &= \frac{|\boldsymbol{r}|}{\sqrt{1+\mu^2}} \left(\boldsymbol{m}_i + \mu \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \boldsymbol{m}_i \right) \\ &= \frac{|\boldsymbol{r}|}{\sqrt{1+\mu^2}} \begin{pmatrix} 1 & -\mu \\ \mu & 1 \end{pmatrix} \boldsymbol{m}_i \end{aligned} \tag{18}$$

Hence,

$$\boldsymbol{m}_i = \frac{1}{|\boldsymbol{r}|\sqrt{1+\mu^2}} \begin{pmatrix} 1 & \mu \\ -\mu & 1 \end{pmatrix} \boldsymbol{r} \tag{19}$$

Plugging this back in Equation 3, we get

$$\begin{aligned} \boldsymbol{r} \cdot \boldsymbol{m}_i - d_i &= \frac{1}{|\boldsymbol{r}|\sqrt{1+\mu^2}} \boldsymbol{r}^T \begin{pmatrix} 1 & \mu \\ -\mu & 1 \end{pmatrix} \boldsymbol{r} - d_i \\ &= \frac{|\boldsymbol{r}|}{\sqrt{1+\mu^2}} - d_i = 0. \end{aligned} \tag{20}$$

$$\tag{21}$$

Since $\boldsymbol{r}$ is a linear function of t, this is a quadratic equation in time and can be solved for t.

8 Collision Detection in Configuration Space

We now describe the collision detection algorithms we implemented. These algorithms are essentially a specialization to $\Re^2 \times S^1$ of the six DOF collision avoidance algorithms of [Don 84, 87]. The basics of this geometric representation may be traced to [LoP, BLP]; in particular, see [BLP, Don 87] for a discussion of applicability constraints.

The special structure of the C-functions f_i permit us to define combinatorially precise algorithms for collision detection, and for solving general intersection problems. In particular (see, eg., [BLP] and Section 5) the general form of a C-function in our application is

$$A_1xS + A_2xC + A_3yS + A_4yC + A_5S + A_6C + A_7x + A_8y + A_9 \tag{22}$$

where $\boldsymbol{x} = (x, y)$, $C = \cos\theta$ and $S = \sin\theta$.

Now, as is well-known, we can make the substitution $u = \tan\frac{\theta}{2}$ in (22). Since $S = \frac{2u}{1+u^2}$ and $C = \frac{1-u^2}{1+u^2}$, we obtain a form of the csurface (22) which is quadratic in u:

$$B_1u^2 + B_2u + B_3 = 0 \tag{23}$$

where the coefficients B_i are all affine in x and y. When the root position $\boldsymbol{x}$ is parameterized by eq. (2), then note that the B_i are all affine in the time t, in the initial position of the root $\boldsymbol{p}_0$, and in the root velocity vector, $\dot{\boldsymbol{p}}$.

Pure translational collision detection is straightforward; see, eg., [LoP, Don 87].

8.1 Computing the "Snap:" Pure Rotational Collision Detection

For pure rotational collision detection at a fixed translation $\boldsymbol{x}$, we implement the following algorithm. First, note that we must solve (23) for its u-zeros; these zeros determine the orientation at which the pawl can cross the boundary of the Cspace obstacle CO defined by (5). Each u-zero can be found by solving a quadratic (23). For each of these events, we construct an *intersection* which is a tuple $(\theta, f_i, \mathcal{A}_i, cfamily(A, B))$. These intersections are sorted around the unit circle (by θ). The intersections are traversed in order, and the first valid one is returned. A valid intersection is one where:

Intersection Validity Tests

1. The constraint is zero: $f_i(\boldsymbol{x}, \theta) = 0$. (This is true by root finding).
2. The constraint is applicable: $\theta \in \mathcal{A}_i$.
3. The intersection configuration is on the boundary of the CO (5). For all applicable f_j at orientation θ in $cfamily(A, B)$, $f_j(\boldsymbol{x}, \theta) \leq 0$. Thus, we must test the signs of all other applicable constraints f_j in f_i's family.

If A and B have size m_a and m_b respectively, this algorithm runs in time $O(m_a^2 m_b^2)$. We can achieve a better bound of $O(m_a m_b)$ by employing the algorithm of[9] [Don 87]. Letting $n = m_a m_b$, we can thereby achieve an overall exact simulatiom algorithm that runs in time $O(n^2 \log n)$. We provide a much faster algorithm in section 9 which asymptotically beats this approach by almost a linear factor of n.

Finally, since one C-family is generated for each convex-convex pawl-obstacle pair, this rotational intersection test must be iterated for each C-family, and the minimum valid intersection returned.

[9] We have given the slower algorithm here, because we have found that while it has higher asymptotic complexity it is more robust geometrically and stable numerically. This is probably because of the redundant information in the representation of a C-family.

8.2 Collision Detection Subject to a Holonomic Constraint

A more complicated intersection problem arises when the pawl is sliding on a surface (23), subject to (2) (which specifies the motion of the root). Hence, we view the situation as follows. Suppose the pawl is sliding subject to a type B (vertex-edge) or type A (edge-vertex) constraint. This corresponds exactly to the reference point sliding on a quadric Cspace surface $\ker f_i$ which is generated by those contact features. The constraint on position (2) defines a "plane" in configuration space. The intersection of this plane and $\ker f_i$ form a quadratic curve in Cspace, parameterized by time. Using this parameterization, we can generate the configuration of the pawl at any time; that is, we have solved for how the pawl moves subject to the root motion (2) and subject to the Csurface (23). We must do this for each pawl (since, as in fig. 4, it is possible for both pawls to be in simultaneous contact with different features of the environment).

The first problem we must solve is how to detect when the quadratic path above intersects another csurface. At that point, the new constraint may take over, or the pawl may snap off due to incompatible kinematic constraints.

Each Csurface has form (23), where the coefficients B_i are affine in $\boldsymbol{x}$, and hence affine in t, $\boldsymbol{p}_o$, and $\dot{\boldsymbol{p}}$ (see (2)). We call this a *Trigonometric Quadratic Form (TQF)* [Don 84,87]. To solve for the simultaneous intersection of two Csurfaces in TQF, we view them as simultaneously quadratic in u and affine in t. We treat the variable t as indeterminate, and use the resultant to obtain a quartic in t. We solve for the t-roots, and back-substitute for the u-roots. Naturally, we must check for the degenerate case where the leading coefficients of the TQF's are zero (however, see sec. 8.4.1). Finally, given the (t, u)-roots, we must then perform the intersection validity tests described above in sec. 8.1 (or the faster test in [Don 87]). While there exist closed-form solutions to quartics, we can also solve them numerically using Ferrari's method.

In general, it is very hard to find exact, algebraic procedures for predicting motion subject to a holonomic constraint. This is because most dynamical systems do not lend themselves to such solutions. Our formulation of rotational compliance enjoys purely algebraic solution trajectories that can be computed using simple algorithms, and that require no integration. Of course, this is largely due to the simplicity of the dynamic model. Nevertheless, from the examples in sec. 3, it is clear that it is capable of producing complex behaviors.

8.3 Choosing the Correct Branch

This section deals with the following difficulties. When we reduce motion prediction problems to the existential theory of the real numbers, we obtain our answers by solving polynomial equations. These equations may have multiple roots. However, our system only has one evolution. We must have a method for choosing which root is correct. Conversely, situations will arise where the algebraic equations fail to have roots. We must handle these cases also.

The situation of multiple roots corresponds precisely to the existence of several "statically" or "kinematically feasible" solutions to the dynamical system, only one of which is feasible (or reachable) from the initial conditions. We present a method for choosing the correct root. When the correct root is parameterized by time, it sweeps out a connected component called a "branch." This occurs in our case, where the formal coefficients are

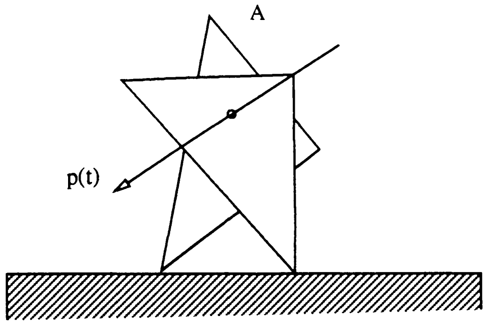


Figure 11: There are two orientations (u-roots) at which the C-function f_i generated by (v_i, e) is zero. Which of these two solutions should we pick?

parameterized by t.

The absence of roots corresponds precisely to the pawl breaking contact with a Csurface, or jamming on the Csurface so that no further motion is possible. We show how to determine which of these events has occurred. These cases are important to handle in practice, and our methods are both theoretically well-founded and implementable.

Consider fig. 11. Clearly, when we solve a TQF such as (23) for its u-roots, we are solving a quadratic; hence we obtain two roots. Now, often one root will be inapplicable, or not on the boundary of the cspace obstacle CO (eq. (5)). However, in fig. 11, both are good roots. Which one should we pick? This is an important question for an implementation. First, we note that even though both are good roots, there is no realizable path between them (and so they are disconnected). Consider the problem of generating a path in cspace subject to csurface constraint (23) and to the straight-line motion (2). (This corresponds to intersecting a quadric csurface with a plane to obtain a quadratic curve, parameterized by time). The path is constructed using elimination methods; consider the problem of "following" this path. One reason we need a systematic way of choosing the correct root is that, due to numerical errors, often one root will appear to become inapplicable "too early" (see, for example, sec 8.4.2), or else, a root appears to just slide off the boundary of the cspace obstacle. At this point the implementation finds the other root (as in fig. 11). This root is applicable, and on the boundary, and so the predicted path "jumps" discontinuously from one root to the other, even though no such path is physically feasible.

This example illustrates pathological behavior. Fortunately, we can give a principled way for choosing the correct root. Essentially, one is finding zeros of a simple algebraic variety in the plane; the problem is equivalent to choosing the correct "branch" of the variety. We show how to do this algorithmically. In the process, we also derive a method for determining when a pawl "jams" against a constraint, and when a pawl "snaps off" a constraint due to incompatible differential kinematics. The two prove to be related problems.

Let us assume that the Csurface is in the form (from (23))

$$S(u,t) = A(t)u^2 + B(t)u + C(t) = 0 \tag{24}$$

where

$$
\begin{aligned}
A(t) &= A_0 + A_1 t \\
B(t) &= B_0 + B_1 t \\
C(t) &= C_0 + C_1 t
\end{aligned}
\tag{25}
$$

and $u = \tan\frac{\theta}{2}$ as usual. Then the branch problem is essentially the problem of choosing the correct sign of the radical in

$$
u = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}. \tag{26}
$$

The problem of determining when the pawl snaps off or jams is essentially that of determining if the discriminant $\Delta(t) = B^2 - 4AC$ is becoming negative, and hence failing to have a real solution. Let us discuss this second case first, and then we will return to discuss the branch problem (how to choose the sign of $\sqrt{B^2 - 4AC}$) afterwards.

8.3.1 Application: Snapping Off and Jamming

See fig. 12, plotting the discriminant $\Delta(t) = B^2 - 4AC$ vs. time, for type (B) and (A) constraints. We illustrate the qualitative contact regions in the space. In the type (B) case, the "pawl" here is drawn as a straight line segment, pivoting about its end point. (The pivot is the center of rotation of the pawl—where it would be attached to a root body). The other endpoint is in contact with an edge. A type (B) constraint is a csurface generated by a vertex of the moving pawl and an edge of the obstacle. A type (A) constraint is generated by an edge of the moving pawl and a vertex of the obstacle. Hence in the type (A) figure, the edge moves with the pivot $\boldsymbol{p}$ and the vertex v is stationary, whereas in the type (B) figure, the edge is stationary, and the vertex at the end of the line segment pawl moves.

Now, the discriminant $\Delta(t)$ is non-negative if and only if eq. (24) has a real solution. hence we need the following test for the zero crossings of the discriminant $\Delta(t)$, which is obtained by solving the quadratic $\Delta(t) = 0$ for its root t_z. That is,

$$
\begin{aligned}
\Delta(t) &= B^2 - 4AC \\
&= (B_1^2 - 4A_1C_1)t^2 + 2(B_0B_1 - 2A_0C_1 - 2A_1C_0)t + B_0^2 - 4A_0C_0.
\end{aligned}
\tag{27}
$$

This will produce two times t_{z_1} and t_{z_2}, with $t_{z_1} \leq t_{z_2}$. Clearly, the time of the first collision with this surface, t_c, must satisfy $t_{z_1} \leq t_c \leq t_{z_2}$ for type (B) csurfaces, and $t_{z_1} \geq t_c$ or $t_c \geq t_{z_2}$ for type (A). Therefore, the *next* critical time t_{z_n} is $t_{z_n} = t_{z_2}$ for type (B) csurfaces. For type (A) csurfaces, $t_{z_n} = t_{z_1}$ if $t_c \leq t_{z_1}$. Otherwise, $t_{z_n} = \infty$ if $t_c \geq t_{z_2}$ (i.e., there is no critical time).

If t_{z_n} is finite, we must now check the behavior at this time. Two things can happen as t is increased a small amount from t_{z_n}. See fig. 13. The pawl-obstacle contact is represented as in fig. 12. For either type (B) or (A) contact, we can either (i) break contact or (ii) jam. The figure shows these possibilities.

Now, which behavior occurs can be determined from the Csurface equation. We assume that $S(u, t) > 0$ means that the vertex is outside the half-space bounded by the edge.

Equation (24) for $S(u, t)$ can be rewritten as

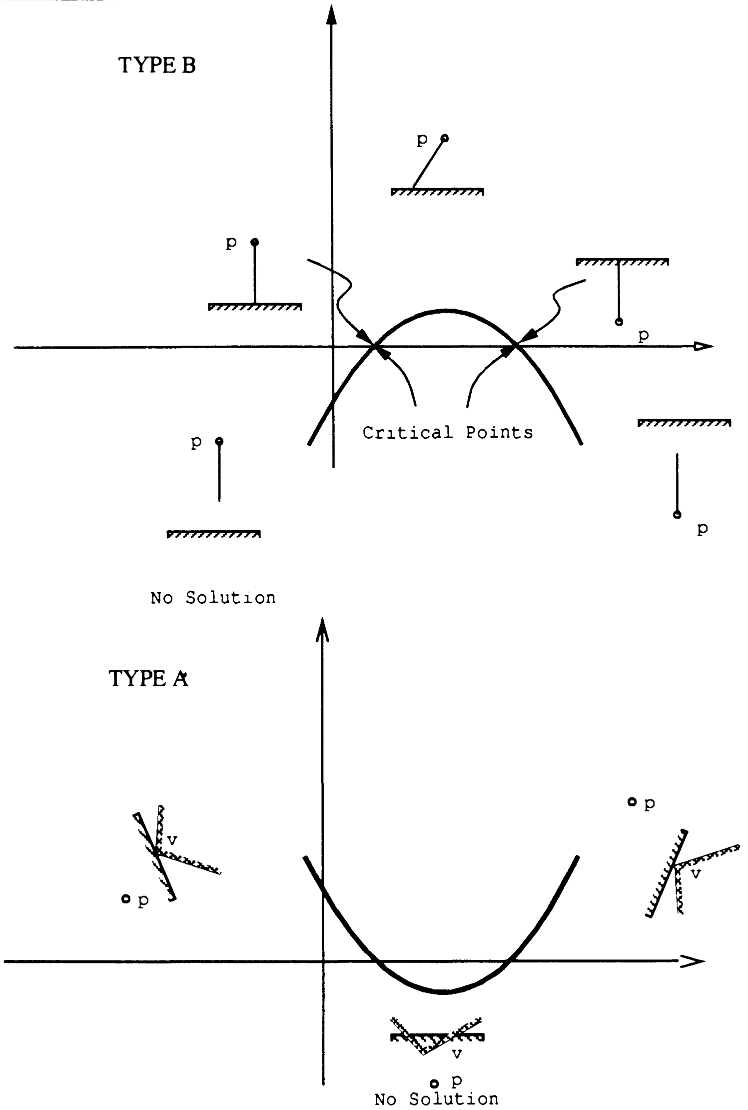


Figure 12: Plotting the discriminant $\Delta(t) = B^2 - 4AC$ vs. time, for type (B) and (A) constraints. We illustrate the qualitative regions in the space. The "pawl" here is represented as follows. For a type (B) constraint, the pawl is a straight line connecting the pivot $\boldsymbol{p}$ with the contact vertex. The obstacle edge is stationary. For a type (A) constraint, the edge moves rigidly with $\boldsymbol{p}$, and v is a stationary obstacle vertex.

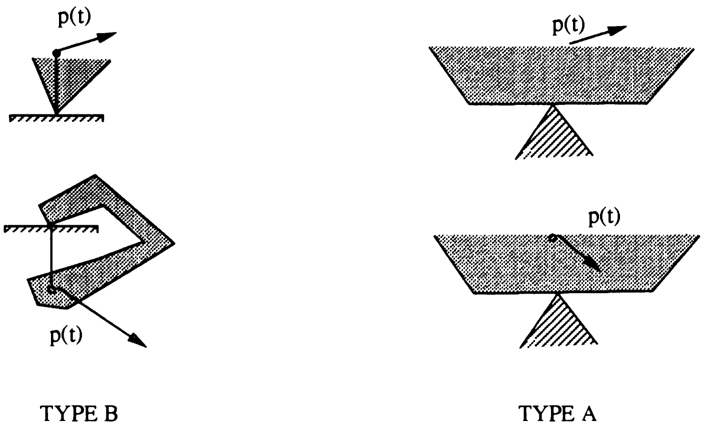


Figure 13: If t_{z_n} is finite, we must now check the behavior at this time. Two things can happen as t is increased a small amount from t_{z_n}. We can break contact (i) or jam (ii). See fig. 12. $\boldsymbol{p}$ is the pivot point, and $\dot{\boldsymbol{p}}$ the pivot velocity. Type (B) and (A) contacts are shown. Case (i-B) (break contact) shows the friction cone. Case (i-A) is impossible, and cannot occur unless we started from this configuration.

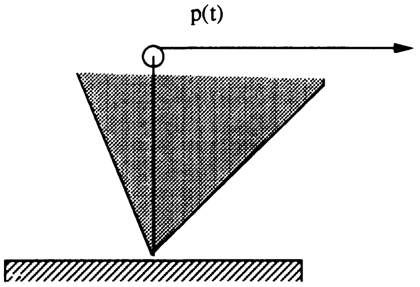


Figure 14: Non-generic case of translation parallel to the edge.

$$S(u,t) = (A_1u^2 + B_1u + C_1)t + (A_0u^2 + B_0u + C_0) \tag{28}$$

First, we solve for u_z at time t_{z_n} from eq. (24). There should be only one solution:

$$u_z = \frac{-B(t_{z_n})}{2A(t_{z_n})} = \frac{-(B_0 + B_1t_{z_n})}{2(A_0 + A_1t_{z_n})}. \tag{29}$$

Next, we examine the sign of the coefficient of t in eq. (24) for this value of u_z. Let $\sigma = A_1u_z^2 + B_1u_z + C_1$.

1. If $\sigma \geq 0$, then the pawl breaks free. It performs a "snap" (pure rotation) towards the zero position.

2. Else, $\sigma < 0$. The pawl is jammed.

3. Note: $\sigma = 0$ is the non-generic case of translating parallel to the edge (see fig. 14). This should probably be considered a "snap", but the exact semantics are somewhat unclear.

8.3.2 The Branch Problem

We now return to the branch problem. This is essentially the problem of choosing the correct sign of the radical in eq. (26). This may be done as follows. First we define the following algorithm, which computes the branch sign for a Csurface at time t and orientation u.

Algorithm Branch-Sign(u, t, Csurface)

1. *Let* $A(t)$, $B(t)$, $C(t)$ *be the coefficients of the Csurface.*
2. $x \leftarrow -\frac{B(t)}{2A(t)}$.
3. $y \leftarrow \frac{1}{2A(t)}\sqrt{B^2(t) - 4A(t)C(t)}$.
4. *If* $u = x + y$ *then return* Plus.[10]
5. *If* $u = x - y$ *then return* Minus.

Now, every time we move onto a new Csurface, we record its sign *Branch-Sign*(u_c, t_c, *Csurface1*). When new candidate intersections are produced when sliding along *csurface1*, then we check that the corresponding u_{new} and t_{new} for the intersection fall on the branch of *csurface1* with the same branch-sign. That is, we check whether or not *Branch-Sign*(u_c, t_c, *Csurface1*) = *Branch-Sign*(u_{new}, t_{new}, *Csurface1*).

8.4 The Genericity of Intersection Problems

Many algebraic robotics and motion planning algorithms make certain "genericity" or general position assumptions about the systems of polynomials they manipulate. We now discuss how certain of these assumptions are not necessary in our algorithm, because (in one sense), our constraints are "never singular:" (we make this notion precise below). On the other hand, there are other general position assumptions which would seem safe (and are, in fact, common in geometry) which, it turns out, are *not* valid in our application. We call this situation "forced non-genericity." Intuitively, it occurs with rotational compliance because of the tendency of the manipulated parts to rotate until many features are simultaneously aligned. In such cases, special techniques are needed to solve the non-generic intersection problems.

[10]With finite precision arithmetic, these equality checks cannot be exact.

Figure 15: The geometry of the flexible object and the environment are the same as in fig. 4. However, the assembly plan is to move diagonally in direction $(1, -\frac{4}{10})$ instead of straight down. The bottom vertex of the left pawl hits first. The next segment of the motion is subject to this constraint, in 15b. At the end of 15b, both pawl vertices lie on the edge. In 15c-d, the pawl is dragged over the top of the "T". In 15f-g, it snaps off to the rest position.

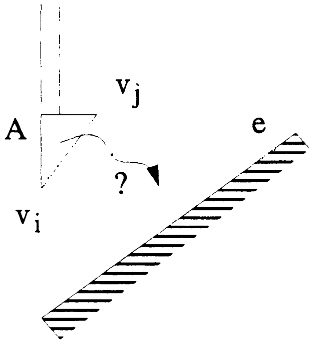


Figure 16: A detail of the bottom of the pawl from fig. 15.

8.4.1 The Inherent Genericity of C-functions in Trigonometric Quadratic Form

In essence, our model permits a reduction of the motion prediction problem to solving polynomial systems of equations. Now, for arbitrary polynomials the leading coefficients of these systems can vanish. This singular situation is undesirable, since in these cases the resultant gives us no information, necessitating special checks. However, in the special case of TQFs that arise from C-functions, it turns out that we do not have to check explicitly for this case. More specifically, the degenerate case of the formal leading coefficients vanishing does occur. But if this happens, it means that in the case of TQFs, $\theta = \pi \pmod{2\pi}$ is a solution for the trigonometric equations [Pai88]. This means that for Csurfaces in TQF, we can blindly apply elimination theory (and especially, the theory of resultants) to solve the polynomial systems. This fortuitous circumstance is only true in our special application, where the quadric surfaces come from TQFs like (23). Thus, while many algebraic algorithms require the system of polynomials to be in general position (this is called a genericity assumption), our method has no such genericity requirement.

8.4.2 On the Forced Non-Genericity of Intersection Problems Subject to a Holonomic Constraint

Whereas many algebraic motion planning algorithms can make assumptions about general position (or genericity) of the polynomial constraints, when we predict motions subject to a holonomic constraint, we find that these assumptions may not hold, and that we are forced to solve non-generic intersection problems.

See fig. 15. The geometry of the flexible object and the environment are the same as in fig. 4. However, the assembly plan is to move diagonally in direction $(1, -\frac{4}{10})$ instead of

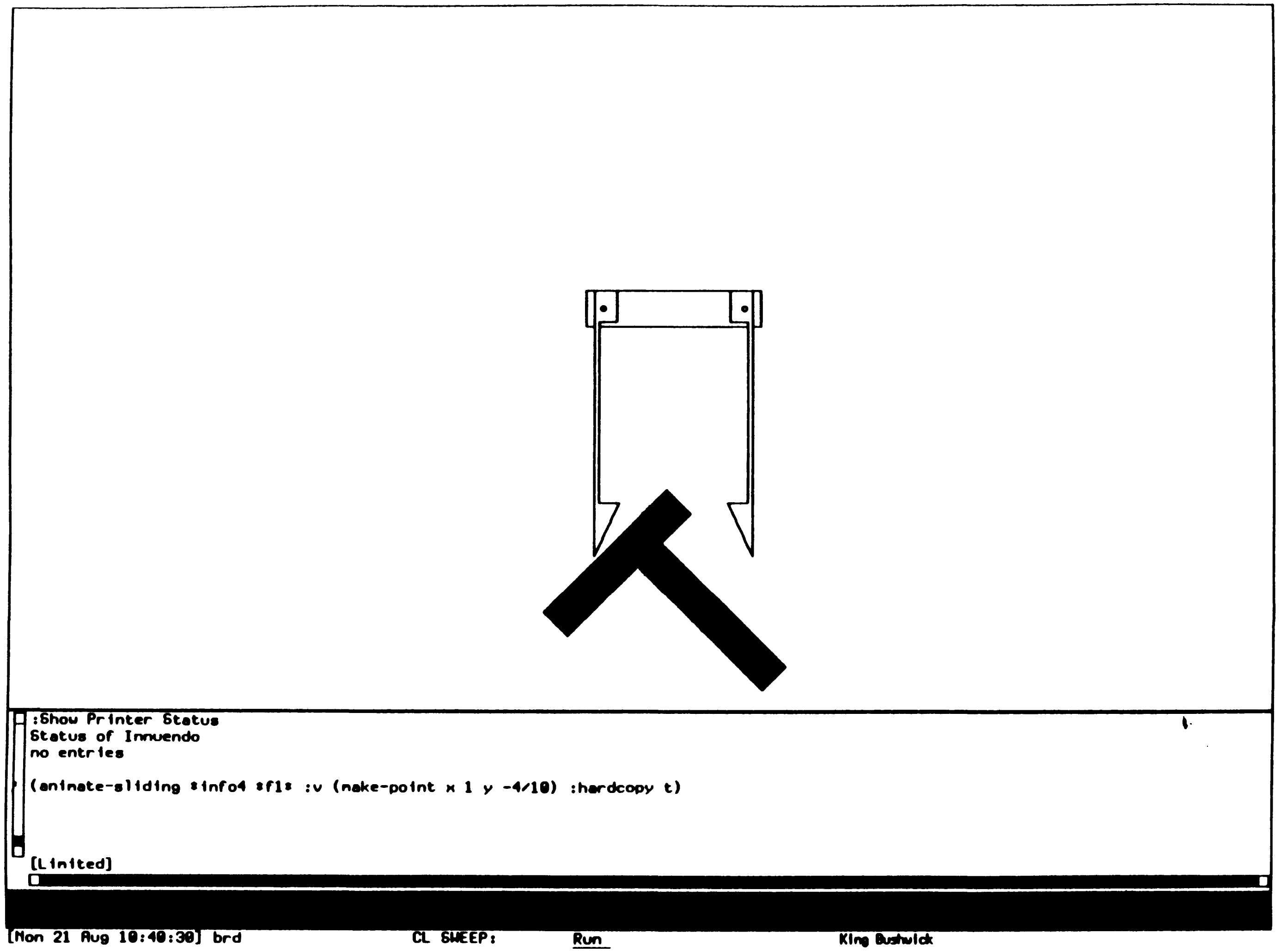

:Show Printer Status
Status of Innuendo
no entries
(animate-sliding *info4 *f1* :v (make-point x 1 y -4/10) :hardcopy t)
[Limited]
[Mon 21 Aug 10:40:30] brd
CL SWEEP:
Run
King Bushwick

Figure 15a

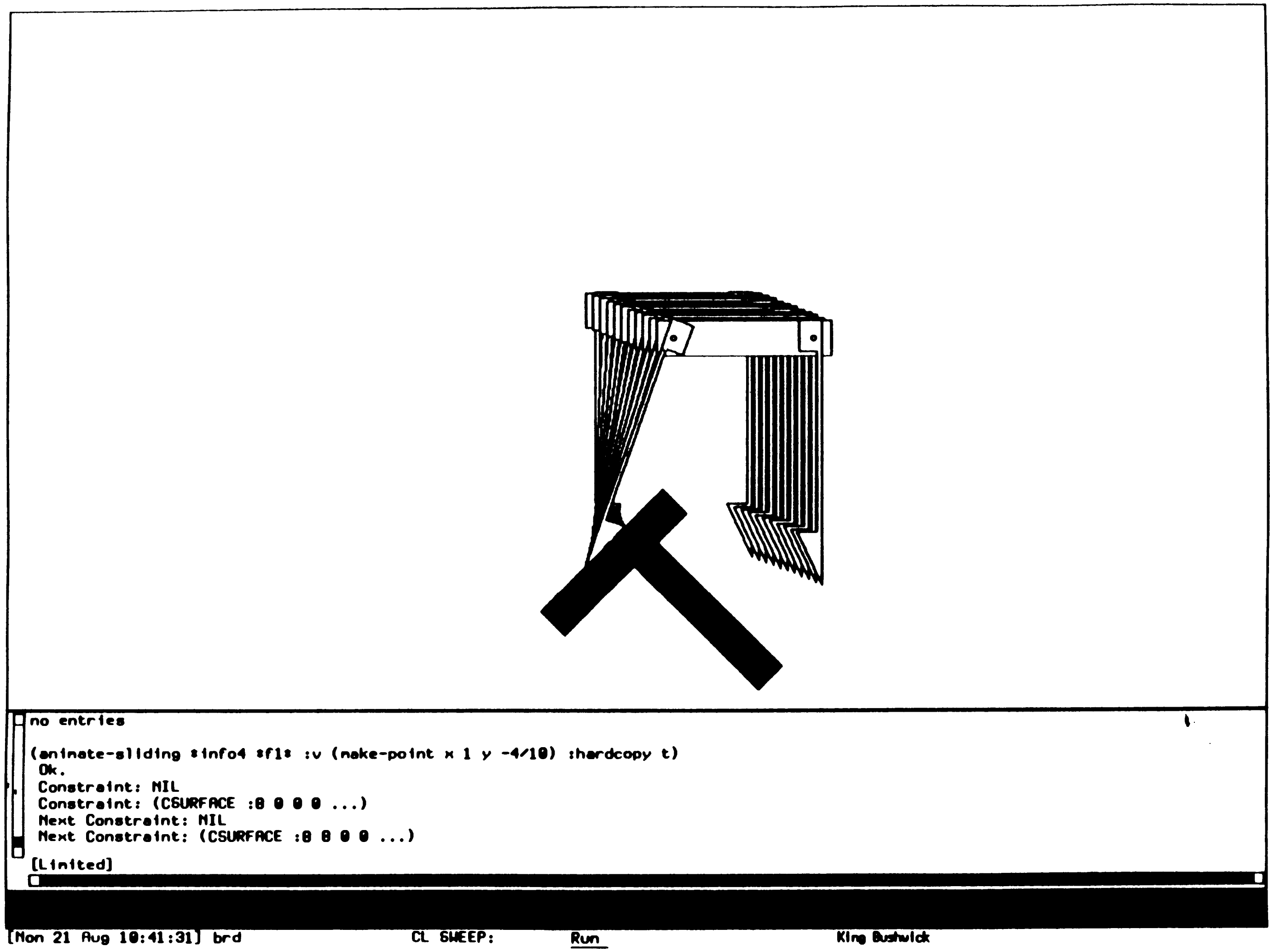

no entries
(animate-sliding *info4 *f1* :v (make-point x 1 y -4/10) :hardcopy t)
Ok.
Constraint: NIL
Constraint: (CSURFACE :0 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 ...)
[Limited]
[Mon 21 Aug 10:41:31] brd
CL SWEEP:
Run
King Bushwick

Figure 15b

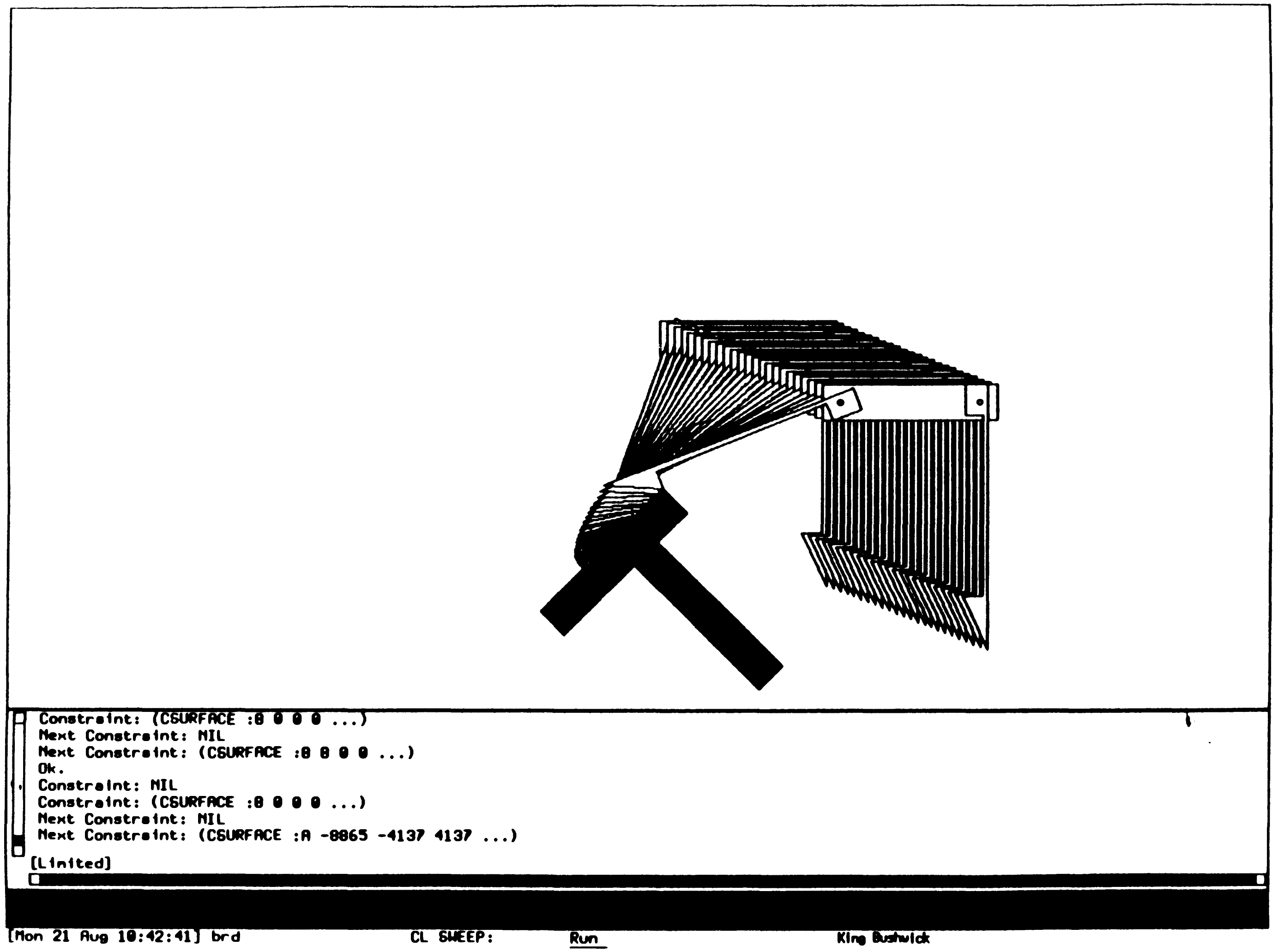

Constraint: (CSURFACE :0 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :0 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -8865 -4137 4137 ...)
[Limited]
[Mon 21 Aug 10:42:41] brd
CL SWEEP:
Run
King Bushwick

Figure 15c

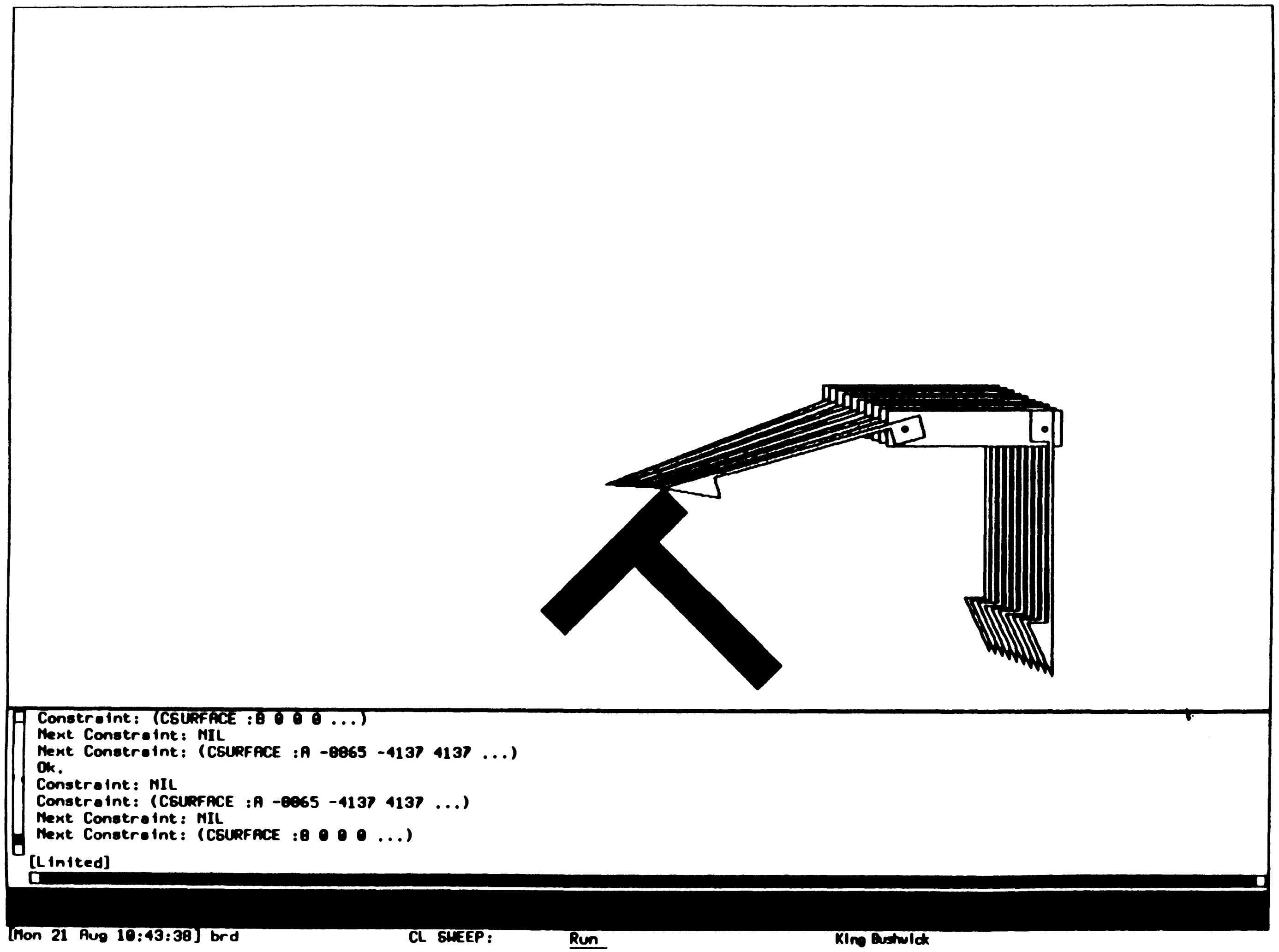

Constraint: (CSURFACE :B 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :A -8065 -4137 4137 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :A -8065 -4137 4137 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :B 0 0 0 ...)
[Linited]
[Mon 21 Aug 10:43:38] brd
CL SWEEP:
Run
King Bushwick

Figure 15d

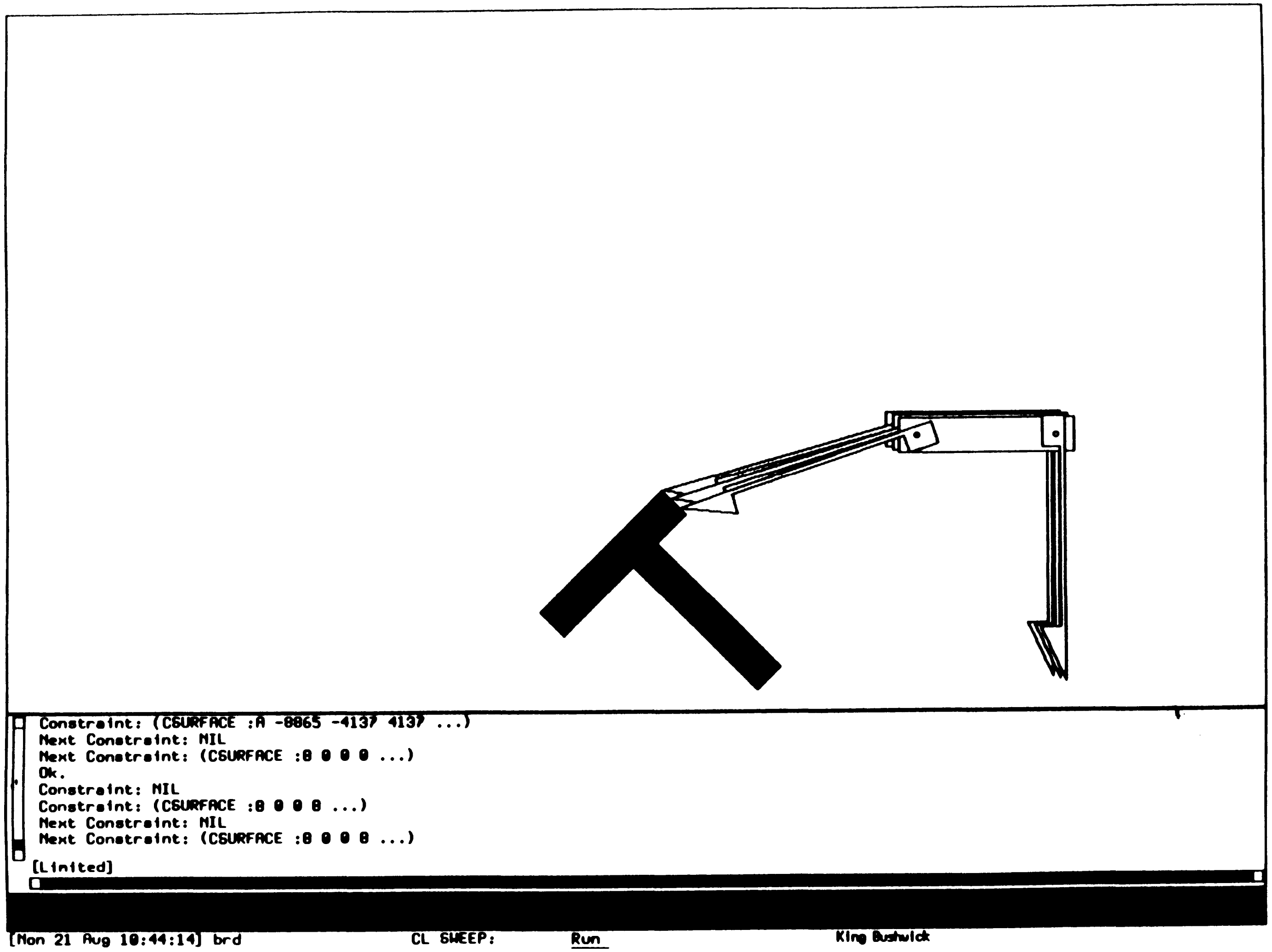
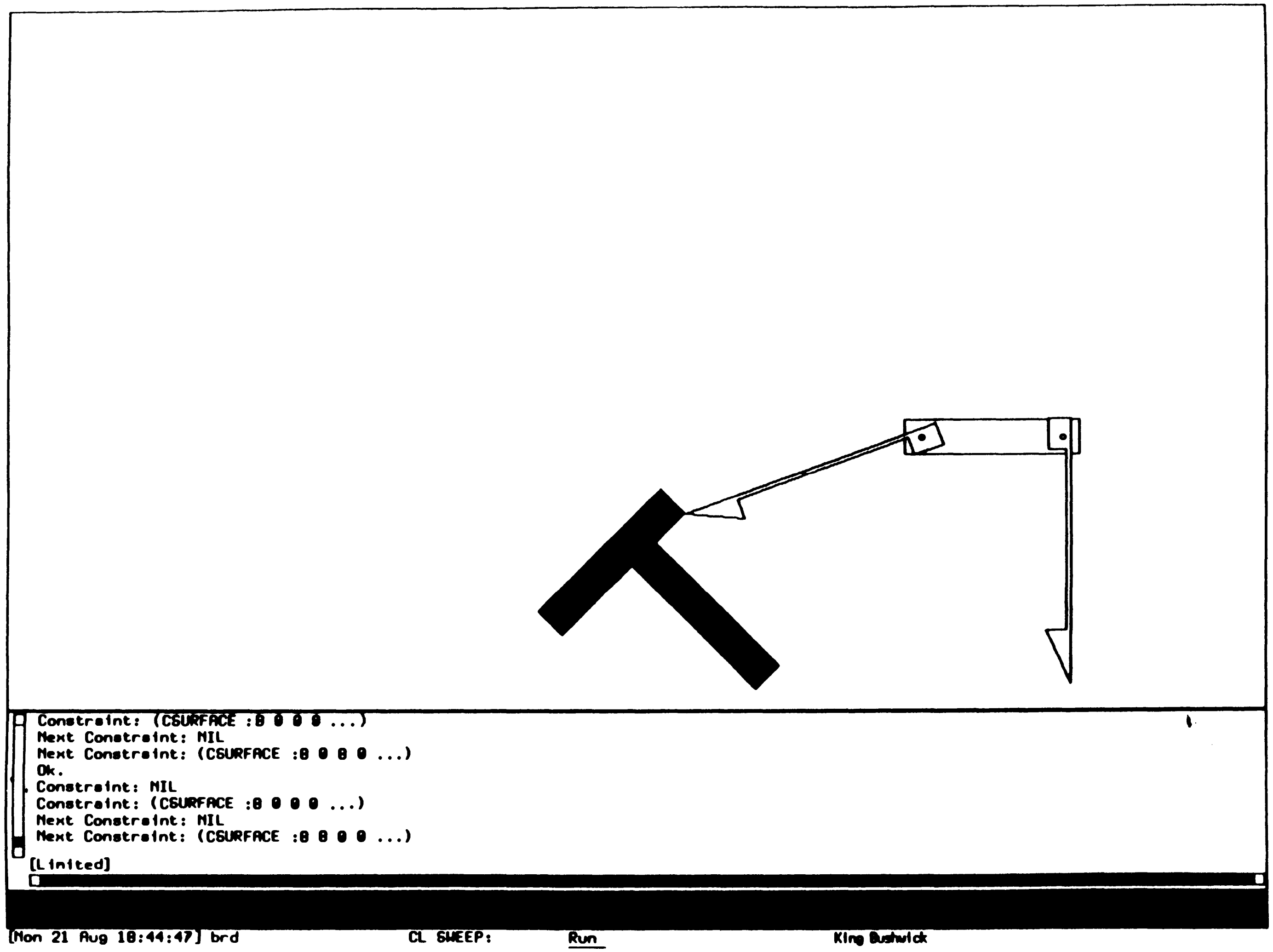

Constraint: (CSURFACE :A -8865 -4137 4137 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :0 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 ...)
[Limited]
[Mon 21 Aug 10:44:14] brd
CL SWEEP:
Run
King Bushwick

Figure 15e



Constraint: (CSURFACE :0 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 ...)
Ok.
Constraint: NIL
Constraint: (CSURFACE :0 0 0 0 ...)
Next Constraint: NIL
Next Constraint: (CSURFACE :0 0 0 0 ...)
[Limited]
[Mon 21 Aug 10:44:47] brd
CL SWEEP:
Run
King Bushwick

Figure 15f

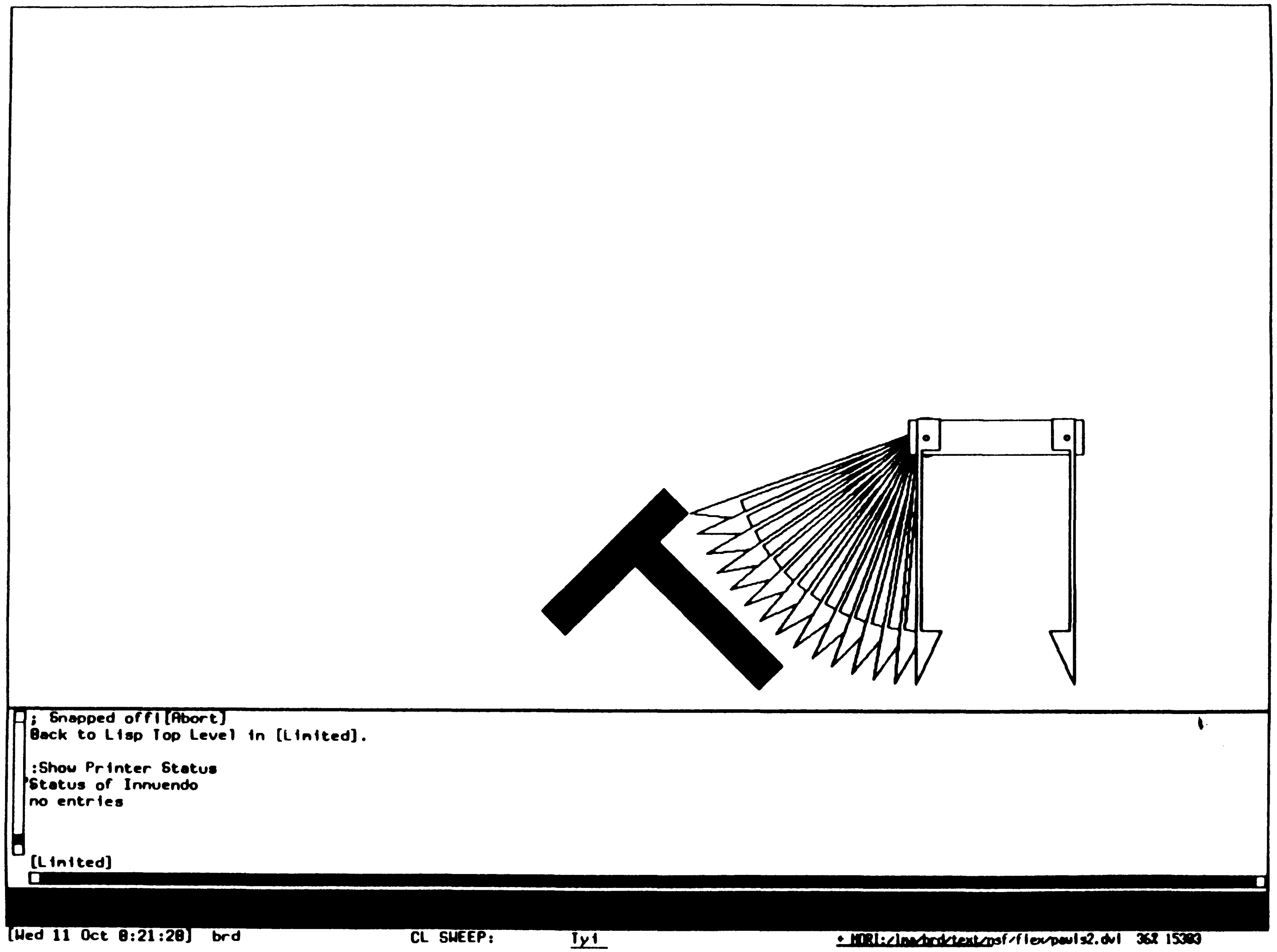

; Snapped off! [Abort]
Back to Lisp Top Level in [Linited].
:Show Printer Status
Status of Innuendo
no entries
[Linited]
[Wed 11 Oct 8:21:28] brd
CL SWEEP:
Tyi

Figure 15g

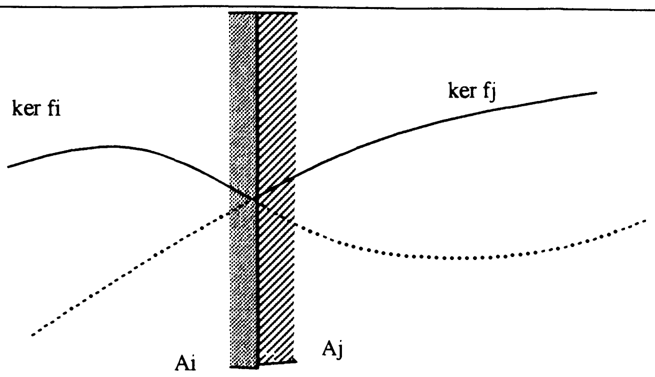


Figure 17: $\ker f_i$ is the csurface for contact (v_i, e) (see fig. 16). $\ker f_j$ is the csurface for contact (v_j, e). The rotationally compliant path ϕ is following $\ker f_i$, when it simultaneously (i) strikes the boundary of f_i's applicability region $\mathcal{A}_i$, (ii) pierces the boundary of f_j's applicability region $\mathcal{A}_j$, and (iii) hits the zero-set of f_j. ϕ is shown schematically in solid lines.

straight down. See fig. 16 for a detail of the bottom of the pawl. The bottom vertex v_i of the left pawl hits the obstacle edge e first. The next segment of the motion is subject to this constraint. That is, the motion is compliant subject to the csurface $\ker f_i$ generated by (v_i, e). Now, refer fig 16 again, and consider an *arbitrary* (generic) motion of the triangle (a path in Cspace). Obviously, this path can cause either constraint (v_i, e) or (v_j, e) to be violated. However, we say that *generically*, both will not be violated at the same time. That is, for a path $\phi : [0,1] \to C$, it will be generically true that $f_i(\phi(t))$ or $f_j(\phi(t))$ is non-zero. (f_j is generated by (v_j, e)). This would be a good general position assumption, but unfortunately, it is simply not true for collision detection subject to a rotationally compliant motion constraint. In fig. 15, we see that as the motion evolves, A rotates about v_i as v_i slides on e. Eventually, this rotation brings v_j down on e. In fact, constraint f_i expires (that is, we pass out of its applicability region) at the precise time that f_j is activated (we pass into its applicability region); at the same time, f_j changes sign from positive to zero.[11] More precisely, at this time t, the orientation $\theta(t)$ crosses the boundary of $\mathcal{A}_i$ and $\mathcal{A}_j$ (which share an endpoint), and $\phi(t)$ hits the zero set of f_j. See fig. 17.

The non-genericity illustrated in figs. 15-17 in fact occurs for a large (generic) class of paths and initial conditions. This is not surprising, since of course one of the outcomes of rotational compliance is to align parts. Second, it is not surprising that when one constraint (C-function) expires, it is "replaced" by a constraint with "neighboring generators." This observation has been formalized by [Don 84, 87] and also exploited by [ELP]. For these reasons, naturally, it would seem as if non-generic examples such as fig. 15 would happen all the time with our system. In fact, this has been our experience. It is unfortunate, in that it places high demands on the robustness of our algebra system. In particular, the applicability constraint boundaries of $\mathcal{A}_i$ and $\mathcal{A}_j$ are, in fact, computed as the zeros of polynomials also. Hence, in observing that three events occur simultaneously (fig. 17) we are effectively saying that three polynomials (in t) have simultaneous zeroes. When these zeroes appear to occur at different times because of numerical errors, then consistency is not maintained and errors

[11]We use "expire" and "become active" following [Don 87].

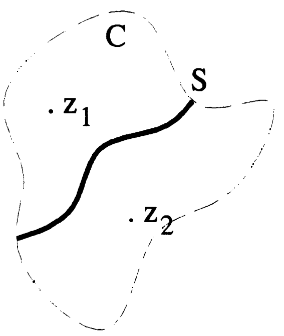


Figure 18: Abstract depiction of forced non-genericity. In a configuration space C consider a lower dimensional algebraic singular set S of "bad" points that we wish to avoid. For example, S could be the set of singular configurations of the forward kinematic map.

can occur. For example, if we find that ϕ crosses $\ker f_j$ before it crosses the applicability boundary $\mathcal{A}_i \cap \mathcal{A}_j$, then that intersection will be judged non-applicable and discarded. If the events are ordered the other way, the intersection is detected.

We can attempt to maintain consistency by introducing additional tests into the algorithm, based on topological information. For example, (see [Don 87]), the polynomial g_{ij} whose zeros define the applicability boundary $\mathcal{A}_i \cap \mathcal{A}_j$ is essentially $f_i - f_j$. Hence, at any given configuration, there are consistency constraints between the signs (and values) of f_i, f_j, and $g_{ij} = f_i - f_j$. Second, we could employ the techniques of [Don87, ELP], which compute, when a constraint "expires", which constraints with neighboring generators become applicable. Thus, when f_i expires at time t, we have $\phi(t) = \mathcal{A}_i \cap \mathcal{A}_j$. We can look at $\dot{\phi}(t)$ and the neighboring generators of v_i and e, to determine that f_j must "replace" f_i in the applicability set (see fig. 16).[12] Finally, we must perform these constraint replacement computations "subject to a holonomic constraint". That is, if f_j replaces f_i in free space, it suffices to simply substitute f_j for f_i in the applicability set. However, in contact, subject to the holonomic constraint

$$f_i(\boldsymbol{x}, \theta) = 0 \tag{30}$$

we must not only update the applicability set, but we must also update the constraint (30). That is, we must simultaneously replace f_i by f_j in the applicability set, and replace (30) by the constraint $f_j = 0$. This constraint replacement mandates that $f_j = 0$. (This is equivalent to the saying that we maintain contact). Hence, we know that at time t, we should simultaneously have f_i, f_j, and g_{ij} vanishing. If because of numerical errors these zeroes occur at different times, then we should in effect "identify" these times into one canonical, simultaneous zero-crossing time for topological consistency.

Finally, we note that the issue of forced non-genericity is subtle mathematically, and has connections to other branches of singularity theory. For example, consider fig. 18. In

[12] The ***applicability set*** is the set of all constraints (C-functions) that are applicable at a configuration.

general, we have a configuration space C containing a lower dimensional singular set S that is algebraic. (In our case, S is the boundaries in fig. 16). Now, if we choose two points z_1 and z_2 randomly in C, the chance that one will lie in S is essentially zero (this is really the definition of measure zero). However, if we now fix z_1 and z_2 and consider all paths from one point to the other, it turns out that "many" of them—a measurable set—cross S. In particular, if S separates C, then one might expect this probability to be roughly proportional to the ratio of the measure of the two components of $C - S$. Finally,

Fact:: *If z_1 and z_2 are disconnected by S in C, then any path from z_1 to z_2 will cross S.*

This means that any such path must experience a singular configuration at some time. This situation is exactly parallel to ours, in which a singularity cannot be avoided. Note that this problem arises in other applications—for example, in [PL92], we observed S as the singular set of a forward kinematic map. By the claim above, this means (for example) that any path from z_1 to z_2 must pass through a singularity.

We have sketched an approach to handling forced non-genericity that exploits the specific characteristics of our domain. General techniques for handling these problems await further research.

9 A Sweep Algorithm for Simulation

The focus of our work has so far been on modeling and robotics issues. In this section we give a fast algorithm by mounting a computational geometric attack. In addition, the new algorithm sheds light on several interesting combinatorial and structural issues. Furthermore, it leads to a systematic classification of special cases, and has connections to the qualitative analysis of dynamical systems.

We have shown how our model of compliance permits us to obtain algebraic, closed-form solutions to simulation problems for a rotationally compliant object. In particular, the linear map $\boldsymbol{p}(t)$ is given by eq. (1), and once we "rationalize" rotations via the standard substitution $u = \tan\frac{\theta}{2}$, each map ϕ_h is piecewise-cubic. Based on these results, a simple algorithm for simulation is as follows.

Simple Algorithm

1. Given a state x of the system, possibly subject to a configuration space constraint f (f has form eq. (3) or (4)), calculate a local solution trajectory Φ respecting f. Φ will be a linear or cubic curve in the configuration space.

2. The configuration space constraints are algebraic surfaces of bounded degree. Intersect Φ with the surfaces to perform exact collision detection.

3. A collision results either in termination of the motion or in a change of constraint. Update f if necessary.

4. Repeat.

Suppose the obstacles have m_b vertices and the moving object (root and one pawl) has m_a vertices. Let $n = m_a m_b$ be the measure of the geometric complexity. In sec. 8 we were able to show that this simple algorithm runs in time $O(n^2 \log n)$. For k pawls, the complexity of the simple algorithm was $O(kn^2 \log n)$.

In this section, we give a new simulation algorithm that is also exact, and runs in time[13] $O(k\lambda_r(n) \log^2 n)$. Our method reduces the simulation problem (def. 2.1) to a plane sweep of an arrangement of algebraic curves in configuration space. To obtain this result we prove the following points. For simplicity of presentation, assume below that $k = 1$, and hence configuration space $C = \Re^2 \times S^1$.

1. The simulation never leaves a connected component F_0 of free configuration space.

2. The solution trajectory Φ of the system is piecewise cubic in the configuration space, and the dynamics may be reduced to erecting cubic "local" configuration space constraints.

3. Translational motion of the root polygon (eq. (1)) restricts the reachable configurations to a 2D cylinder $Y \subset C$.

4. Y embeds in C as follows:

$$\begin{array}{ccc} Y & \hookrightarrow & \Re^2 \times S^1 \\ (t, \theta) & \mapsto & (\boldsymbol{p}(t), \theta). \end{array} \tag{31}$$

   where t moves along the axis of the cylinder. Hence we view Y as $\Re \times S^1$. The time evolution $t \in T$ of the system corresponds to the $\Re$ factor. Hence the solution "sweeps" along the cylinder Y in the direction of the axis.

5. The configuration space constraints are manifest as cubic curves on Y.

6. As we sweep the cylinder Y, a simple dynamical system models the motion of the configuration point on the configuration space constraints. The orbits of this system are piecewise algebraic of bounded degree.

7. Parameterize Y to the plane (this only takes 2 charts). Then $F = Y \cap F_0$ is defined by a planar arrangement of cubic curves.

8. F has size $O(\lambda_r(n))$ and can be constructed in time $O(\lambda_r(n) \log^2 n)$ using a red-blue merge algorithm, as described in [GSS].

9. We can compute the solution Φ by a plane-sweep of the slice F, i.e., by sweeping a planar arrangement of cubic curves. This allows us to solve the motion prediction problem exactly in $O(\lambda_r(n) \log^2 n)$ overall time.

---

[13]Recall that $\lambda_r(n)$ is the (almost linear) maximum length of (n, r) Davenport Schinzel sequences [GSS] and r is a small constant. For practical purposes, $\lambda_r(n)$ should be regarded as just a tiny bit bigger than $O(n)$. For more on the the theory of Davenport-Schinzel sequences and their applications in computational geometry, see [ASS, HS, KS].

Hence we prove

Theorem 9.1 *The* Simulation Problem *defined above (def. 2.1) can be solved in time* $O(\lambda_r(n)\log^2 n)$ *and space* $O(\lambda_r(n))$, *where* r *is a small constant.*

We outline the proof in the following sections.

9.1 Computing the Connected Component of Free Space

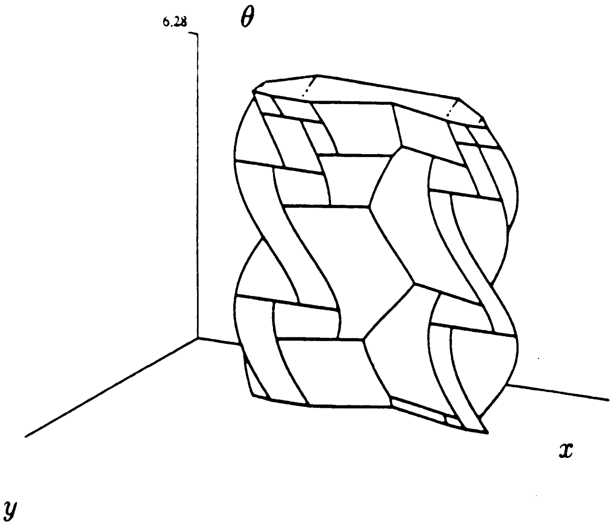
Consider the motion of a single pawl $\mathcal{M}_h$, whose translation is governed by eq. (1). As is well known, if we "rationalize" rotations using the standard substitution $u = \tan\frac{\theta}{2}$, the n constraints given by eqs. (3) and (4) are manifest as algebraic ruled surfaces $\{ f_1, \ldots, f_n \}$ in a 3D configuration space with coordinates (x, y, u). These constraints are simultaneously linear in the position parameters x and y and quadratic in the rotational parameter u. Each surface is only "applicable" for some range of orientations $[u_0, u_1]$, by which we effectively mean that the surface only "exists" for u in this range (see [Don]). See fig. 19, from [Bro].[14]

Figure 19: Configuration space constraints for a moving pawl. (Reprinted courtesy of Randy Brost [Bro]). The sweep plane $L(t)$ intersects the cspace obstacles in a planar arrangement of cubic curves.

Let $\hat{\boldsymbol{u}}$ be a vector along the u-axis (think of $\hat{\boldsymbol{u}}$ as $(0,0,1)$). Now, the constraint of pure translation (eq. (1)) of the hinge point P_h of the pawl $\mathcal{M}_h$ restricts any possible evolution of the system to lie in a "plane" (2D subspace) of (x, y, u)-space (see Figure 19. We call this "plane" P_Y; it has "normal" $\hat{\boldsymbol{u}} \times (\dot{\boldsymbol{p}}, 0)$, and it corresponds to a chart for the cylinder Y discussed above. We note that furthermore, at time t, the state of the system is constrained to lie on a line $L(t)$ parallel to $\hat{\boldsymbol{u}}$ in the plane P_Y. This is the line of points $(\boldsymbol{p}(t), u)$, for $u \in \Re$. The line $L(t)$ sweeps across the plane in direction $(\dot{\boldsymbol{p}}, 0)$ as t increases, and thus we call $L(t)$ the *sweep line.*

Hence, a natural coordinate system for the plane P_Y is given by (t, u). As time t increases, the vertical line $L(t)$ sweeps across P_Y. This line contains the state of the system. It is our task to calculate the u coordinate as t evolves (i.e., increases). Now, P_Y has degree 1 and

[14] We thank R. Brost for providing us with these figures, from [Bro]. See [Bro2] for even nicer pictures.



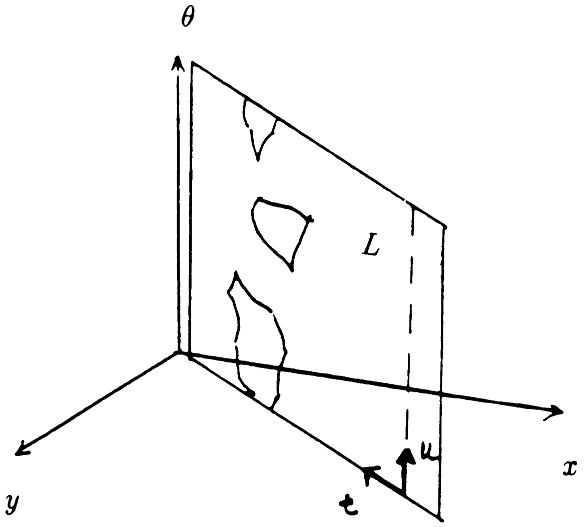


Figure 19

hence when intersected with a constraint f_i we obtain a cubic[15] curve segment γ_i in the (t, u) plane. So all the configuration space constraints are manifest as an arrangement of cubic curve segments $\{\gamma_1, \ldots, \gamma_n\}$ in this plane, where $\gamma_i = f_i \cap P_Y$ $(i = 1, \ldots, n)$. In our algorithm, the sweep line sweeps across this planar arrangement of curves, and as we sweep, we compute the trajectory of the system. "Events" caused by crossing the curves γ_i will modify the trajectory, as we discuss below.

A priori, the arrangement of curves can have complexity $O(n^2)$, and in fact, the set of free configurations in P_Y can also have size $\Omega(n^2)$ in the worst case (see [KS,GSS]). However, we can directly apply the results of [GSS] as follows. We note that the system begins at some configuration $z_0 \in P_Y$. z_0 lies in one connected component $F \subset P_Y$ of free space and the resulting path can never leave F since it corresponds to a physical simulation. [GSS] show that:

Lemma 9.2 [GSS] *The combinatorial complexity of F is $O(\lambda_{s+2}(n))$, and we may precompute it (i.e., compute it before our plane sweep) in time $O(\lambda_{s+2}(n) \log^2 n)$. Here s is a small constant bounding the number of times that two configuration space constraint curves γ_i and γ_j can intersect.*

Recall $\lambda_r(n)$ is the (almost linear) maximum length of (n, r) Davenport Schinzel sequences [GSS], [HS,ASS]. It is very likely that for a wide variety of situations encountered in practice (see, eg., [Bro, Bro2]) that $s \leq 1$. However, it is certain that a worst case bound is $s \leq d^2$ for curves of degree d. Note that $s \leq 1$ would tighten our bound to space $O(n\alpha(n))$ and time $O(n\alpha(n) \log^2 n)$ [HS].

9.2 Sweep Events in Configuration Space Slice

9.2.1 Kinematics

We postpone our discussion of friction until sec. 9.3. Here we treat the frictionless case first, defining six types of local geometric events, called "sweep events" that will be detected and handled during the plane sweep. These events are purely kinematic, i.e., they do not depend on friction.

Having computed the connected component $F \subset P_Y$ containing the initial configuration, we now sweep F with the line $L(t)$ in the (t, u)-plane. By an abuse of notation we will now let γ_i, γ_j etc. denote the curves bounding F that we precomputed (along with their intersections) in sec 9.1. We now define the following sweep events: (i) *translational collision,* (ii) *sliding collision,* (iii) *jamming due to incompatible kinematics.* The sweep algorithm will detect sweep events. Each event will be *handled,* by which we mean that the solution trajectory we compute may be modified. In between events, the trajectory is piecewise algebraic.

First suppose there are no obstacles. Then the trajectory of the system will stay at $u = 0$ in the (t, u)-plane (i.e., the orientation of the pawl will not change). Call the point $z(t)$ on $L(t)$ representing the state of the system the *sweep point.* So $z(t)$ is the solution trajectory we compute.

[15]In fact, each curve γ_i is simultaneously quadratic in u and linear in t.

Now, in the presence of obstacles, sweep events occur as the the sweep line crosses the curves $\{\gamma_i\}$. These events are enumerated in figs. 20-22. We can explain the trajectory computation algorithm like this: the dynamical system described in sec. 2 has the following geometric interpretation in slice P_Y. As the sweep line $L(t)$ crosses the (t, u), plane, the the u-coordinate $u(t)$ of the sweep point $z(t) \in L(t)$ moves. In the plane P_Y, the line $u = 0$ is an *attractor*, and we imagine a vector field on P_Y parallel to the u-axis and pointing towards the t-axis. Hence the attracting vectors are parallel to $-\hat{\boldsymbol{u}}$ for $u > 0$ and parallel to $+\hat{\boldsymbol{u}}$ for $u < 0$. The curves γ_i act as (holonomic) constraints. The sweep point cannot cross these curves, but it can follow them as $L(t)$ moves. They can prevent motion of the sweep point from attaining $u = 0$.

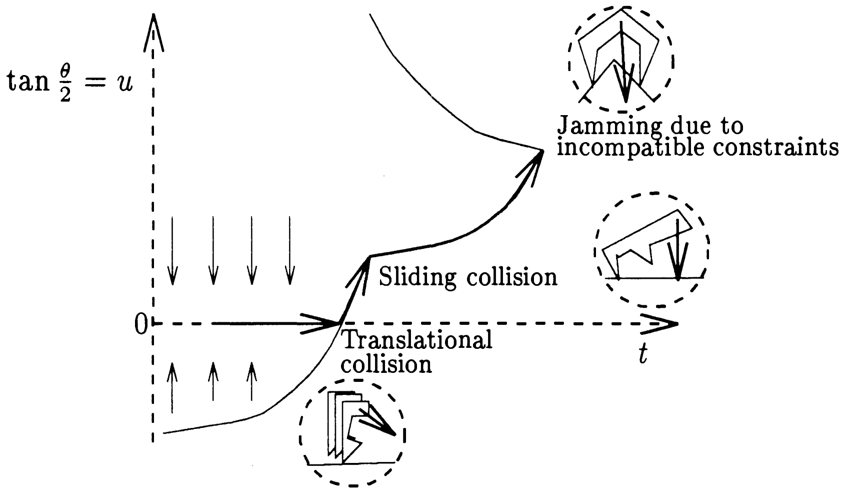


Figure 20: Sweep events: translational collision, sliding collision, and jamming due to incompatible kinematics.

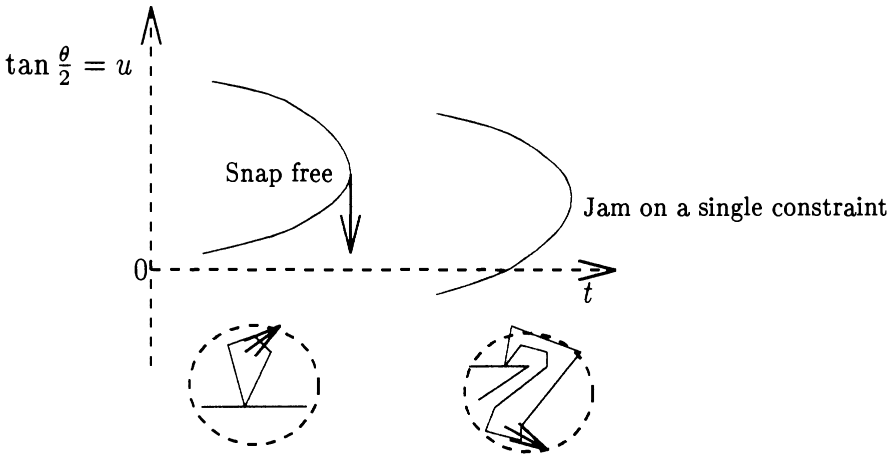


Figure 21: Sweep events: Snapping free and jamming on a single constraints.

For example, see fig. 20. If the trajectory is at the $u = 0$ position and the sweep point $z(t)$ encounters a constraint γ_i, then the sweep point complies to the constraint and is forced to move away from the zero line (u becomes positive here). This corresponds to a pure translational collision, followed by a continued motion of the root which "cocks" the pawl against an obstacle. During this motion, the sweep point follows γ_i. If a new constraint γ_j is reached, then the sweep point slides along the curve γ_j in turn. This corresponds to a sliding collision: while sliding on constraint γ_i, the pawl hits constraint γ_j. The motion continues, following γ_j compliantly. Hence the sliding collision can result in a constraint change. Finally, if the sweep point is following a curve γ_i which crosses $u = 0$, the trajectory breaks contact there and continues along the t-axis. This event is a "dual" subcase of type (i).

As can be seen from fig. 20, some constraint changes result in jamming due to incompatible kinematics. This occurs as follows. Define the outward normal $\boldsymbol{\eta}_i$ of a curve γ_i to point into free space F. Let $\widehat{\boldsymbol{t}}$ be a unit vector in the positive t-direction. Jamming occurs at $\gamma_i \cap \gamma_j$ when both the inner products

$$\boldsymbol{\eta}_i \cdot \widehat{\boldsymbol{t}} \quad \text{and} \quad \boldsymbol{\eta}_j \cdot \widehat{\boldsymbol{t}} \tag{32}$$

are negative. At this point the simulation is terminated, because further motion is impossible.

Pure translational collision events can occur where a curve γ_i intersects the line $u = 0$. *Sliding collisions* can occur when two boundary curves of F intersect, i.e., at $\gamma_i \cap \gamma_j$. Jamming events can occur when both normals at $\gamma_i \cap \gamma_j$ point in the $(-t)$-direction. A non-jamming sliding collision causes a change of constraint (i.e., the sweep point now follows γ_j instead of γ_i). It is clear that sweep events of type (i), (ii), and (iii) are local geometric conditions and can be detected and handled while sweeping the line $L(t)$ over F. Similarly, it is clear that modifying the trajectory $\boldsymbol{z}(t)$ at a sweep event can be done in $O(1)$ time.

9.2.2 Snapping Free or Jamming on A Single Constraint

We now define the sweep events (iv) *snapping free from* and (v) *jamming on a single constraint.* Suppose the sweep point is following a constraint curve γ. A singularity occurs at vertical tangencies of γ. See fig. 21. Assume wlog that γ lies in the halfplane $u > 0$. There are two possibilities. If the F is concave at the singularity, then the sweep point has been following the "upper" branch of the curve. After the singularity, the sweep point follows the vector field attracting it towards $u = 0$. That is, the sweep point moves parallel to the u-axis toward the t-axis. It stops at the first new constraint curve it hits while moving away from the singularity towards the line $u = 0$. If no constraints are encountered, it stops at $u = 0$. This motion corresponds to the pawl "snapping free" from a single constraint edge. It executes an instantaneous pure rotation towards the zero position. If another constraint is in the way, then it stops there.

If F is convex at the singularity, then no further motion is possible, and the motion jams there on a single constraint. At this point the simulation is terminated.

Clearly, singularity (vertical tangency) is a local geometric condition that can be detected during the plane sweep of F, since each curve is algebraic.

There is one more kinematic sweep event that is "dual" to type (iii) jamming due to incompatible kinematics. It is type (vi) *snapping free from a vertex.* It occurs at a constraint change $\gamma_i \cap \gamma_j$ (i.e., the sweep point is following a curve γ_i, and it hits another curve γ_j). However, in this case, both the outward normals $\boldsymbol{\eta}_i$ and $\boldsymbol{\eta}_j$ point in the positive t-direction.

That is, the dot products in eq. (32) are both positive. In this case, the sweep point "snaps free" from $\gamma_i \cap \gamma_j$ and moves vertically towards the attractor $u = 0$. The snapping free happens just as in event (iv) above. Snapping free from a vertex corresponds to the situation where suddenly there are no holonomic constraints on the pawl, so it can move towards its rest position $u = 0$. Conceptually, there is little difference from event (iv) (snapping free from one constraint).

It is clear that sweep events of type (iv), (v), and (vi) are local geometric conditions and can be detected and handled while sweeping the line $L(t)$ over F. It is clear that modifying the trajectory $z(t)$ at a sweep event can be done in $O(1)$ time. To see that six event types suffice, simply enumerate the ways γ_i can (a) intersect γ_j, (b) intersect $u = 0$, or (c) become vertical. Hence we have,

Lemma 9.3 *There are six types of kinematic sweep events, as described above. There are $O(\lambda_{s+2}(n))$ such events overall. Each event is a local geometric condition that can be detected and handled in $O(1)$ time. The output trajectory is piecewise algebraic with at most $O(\lambda_{s+2}(n))$ pieces and degree at most 3.*

Corollary 9.4 *A plane sweep of F that handles all kinematic sweep events can be performed in time $O(\lambda_{s+2}(n) \log \lambda_{s+2}(n))$. This sweep solves the frictionless simulation problem (given F) for a single pawl.*

9.3 Friction

We now briefly describe how friction is handled. From the analysis in 6, the following is clear: for each configuration space surface f_i we can define two constraints g_i and h_i which are also algebraic surfaces of the same degree as f_i. g_i and h_i depend on the direction of assembly $\dot{\boldsymbol{p}}$ in (1).

The surfaces g_i and h_i break up f_i into *sliding* and *sticking* regions. We call these *qualitative dynamical regions (QDR's)*, by analogy with [BD]. In a sliding region, motion is possible as t increases. In a sticking region, equilibrium results, and no further motion is possible (compare work on translational compliant motion, eg, [Don2, Bri]). Now, when we intersect f_i with the plane P_Y to obtain a curve γ_i (see fig. 22) we obtain a 1D slice of these qualitative dynamic regions (sliding and sticking). Now, the Bezout bound gives an a priori $O(1)$ bound on the number of QDR's per surface. However, in fact, the special structure of our constraints ensures that there will be at most 3 QDR's per connected curve γ_i on the boundary of F. Type-B constraints have (at most) one sliding region surrounded by two sticking regions. Type-A constraints have (at most) one sticking region surrounded by two sliding regions.

Now, we define a seventh type of sweep event, (vii) a *sticking event* as follows. Suppose the sweep point is following a curve γ_i. If it enters a sticking region on the curve, then equilibrium is reached and the simulation is terminated. Entry into the sticking region corresponds to crossing another algebraic curve h_i or g_i, and hence is a local geometric event that can be detected and handled during the sweep. Note that g_i and h_i apply only to f_i and do not affect any other surface f_j, and hence we call them *local dynamic constraints*.

Finally we must slightly modify our kinematic plane sweep. After a pure translational or pure rotational collision with a curve γ_i, we first check to see whether we are in a sticking or sliding region on that curve. If it's a sticking region, we terminate the simulation in equilibrium, otherwise we proceed as above (sec. 9). To summarize,

Proposition 9.5 *There are $O(1)$ sticking events per constraint curve γ_i. Each occurs at the intersection of γ_i with a local dynamic constraint (another algebraic curve) in P_Y.*

This completes our proof of the main theorem 9.1.

9.4 Summary

Red-blue merge algorithms allow us to construct a connected component of free space F containing the initial configuration in time $O(\lambda_{s+2}(n)\log^2 n)$. We desire to simulate a simple dynamical system within F. We compute the simulation trajectory using a plane sweep of F. To do this, we augment F with a vector field (defining an attractor at $u = 0$) and we "annotate" each curve γ_i on the boundary of F with certain "markings" at which the dynamical behavior of a sweep point traversing γ_i can change. The markings break the curve into a finite number of subsegments. The markings are: (a) sticking/sliding transition, (b) vertical tangency, and (c) intersection with the line $u = 0$. Each marking is determined by the intersection of γ_i with a line (such as $u = 0$) or a curve (g_i or h_i), or by vertical tangency. Hence each marking is algebraic and there are $O(1)$ markings per curve. Finally, at an endpoint of γ_i we have an intersection with the next cubic curve γ_j on the boundary of F.

10 Conclusions

In this paper, we addressed the problem of predicting the motion of a flexible object amidst a polygonal environment. To this end, we presented a careful analysis of the physics of the interaction of the flexible body with the environment. The analysis yielded several simple tests to determine the motion of the body, which were integrated into an algorithm for predicting the motion under a given motion plan.

This motion prediction problem is the first step in reasoning about such devices. We also described the implementation of the algorithm and related issues, as well the significance of the algorithm for designing compliant parts for ease of assembly. In the appendix B we describe the effect of uncertainty.

Suppose we wish to build a system for reasoning about and analyzing the motion of a "flexible object"—i.e., a compliantly-connected system of rigid bodies. To this end, we developed a basic theory on the motion prediction problem for such bodies, at a fairly abstract level, emphasizing connections to computational mechanics and the long-term behavior of dynamical systems. We discussed how our formulation of the problem led to theoretically precise algorithms for motion prediction with rotational compliance, and showed how these algorithms could be implemented and used to analyze designs for assembly. We stressed the fact that while the theoretical algebraic algorithms we give are correct when exact arithmetic is employed, in practice we must strengthen the algorithms to make them robust when implemented using finite-precision arithmetic. We showed how this may be done in a systematic fashion. Issues of robustness and numerical stability turned out to be related to problems of genericity and branch-choice, and we described the theory behind these problems and their practical solution. We emphasized our computational approach, while considering the pitfalls and subtleties that an implementation foregrounds.

10.1 Summary of Results

The research contributions of this paper include:

1. New algebraic techniques for predicting the motion of objects in contact under our model of rotational compliance.

2. A systematic catalog of singular and non-generic situations that must be handled, and algorithms for dealing with these events.

3. An extension of our algorithm to compute mating force information. This facility can be used to design objects that are easier to mate than to disassemble.

4. An extension of our algorithm to efficiently predict motions given uncertainty in sensing and control (appendix B).

5. A manifesto for the relevance of our approach to engineering, and particularly to design for assembly. As an application, we studied several classes of flexible devices that we term "motion diodes." We developed a classification of these devices into total vs. relative motion "diodes", and described kinematic, frictional, and force diodes. Our algorithm can analyze designs, and classify the diode type. We suggested how motion diodes could be useful in design for assembly.

6. We considered the problem of simulating the motion of compliantly connected rigid bodies in frictional contact with obstacles during an assembly motion. While compliant motion has been considered in a computational geometric setting for pure translations [Don2, Bri, FHS], the problem for rotational compliance has proved resistant to solution. We showed that unlike many simulation problems for rotational bodies, we can obtain exact solutions without integration. We first developed a naïve yet combinatorially precise $O(n^2 \log n)$ algorithm based on collision-detection techniques such as [Don 87].

7. We improve on this algorithm by the introduction of several techniques. The key ideas we use are: red-blue merge algorithms, a simple dynamical systems model, and local dynamic constraints. These tools permit us to reduce the simulation to a plane sweep of a " dynamically annotated" slice of configuration space. More specifically: First, we precompute the connected component of the simulation. This component of free space has low combinatorial complexity (by Davenport-Schinzel arguments) and can be computed efficiently using a red-blue merge algorithm [GSS]. Next, we reduce the simulation problem to a plane sweep of F. To do this, we first introduce additional

local constraints ($O(1)$ per curve bounding F), an attractor at the rest orientation $u = 0$, and a corresponding attractive vector field on F. These constructions allow us to view the plane sweep as a simple dynamical system. This in turn permits us to bound the number of sweep events by $O(\lambda_{s+2}(n))$, which yields our main result. This is one of the first combinatorially efficient, exact solutions to any simulation problem for a rotational mechanical system, or for rotational compliant motion.

10.2 Discussion and Future Work

There are many problems left open for the future. First, we would like to extend our work to "trees" of compliantly connected bodies (articulated bodies). Our flexible objects are trees of depth one; however, one could imagine many compliantly-connected links in a chain. Future work includes the incorporation of more complicated models of the flexible objects (such as that of [TM87]) and the dynamics of interaction. For example, we hope to add more sophisticated dynamics to our work, and to model continuously deformable bodies as well.

Second, we believe our work can be extended to incorporate uncertainty in the initial conditions and in the control. This result would be of considerable interest, since it would permit simulation of a differential inclusion. Exact simulation does not take into account the uncertainty (eg., the impossibility of a comprehensive description of the dynamics of a system), nor error in actuation. We view the introduction of a more realistic mechanical model (rotationally compliant bodies) as a step in this direction. We feel that the key property that allowed us to reduce the local dynamics to a plane sweep is a kind of "monotonicity" that is inherent in our system. It is our hope that other "monotonic" systems (and even differential inclusions) may be simulated using the concept of "simulation as sweep."

We also wish to extend and test our system in analyzing real designs. We are also fabricating parts we have designed, and testing them by assembling them with real robots [Boh92]. Perhaps the most challenging area for the future involves the interaction of our analysis approach with uncertainty. We hope to extend our analysis and algorithms of appendix B to model uncertainty more precisely and to generate designs that can be assembled robustly. In particular, we hope to develop a precise notion of the "stability" of a motion with respect to a design. For example, an assembly plan might be "stable" if the qualitative outcomes of execution are equivalent.

We feel that algorithmic research on design for assembly represents a new direction in robotics which could have some impact on the field. In the future, one could imagine our techniques employed in an approach to *" design as search"*, in which we modify and improve an existing design by changing its geometry incrementally and applying our analysis algorithm. For example, suppose we are given geometric models of two parts to mate, and an approximate "assembly plan" (direction of mating). We envision an algorithm that searches around the boundaries of the models, and modifies the shape by introducing snap-fasteners and ratchet and pawl mechanisms. After each modification, the analysis algorithm described above could be run, to determine how the parts will mate, and the forces required to mate and disassemble them. Depending on the results of the analysis, the design change could be modified, retracted, or declared suitable. Of course, there is a host of conceptual and algorithmic problems that must be solved to make such an approach possible, but we believe that with the algorithmic underpinnings described in our analysis algorithm, a research program in this area is now viable. Such research could help to put design for assembly on a firm algorithmic footing, build software tools for generating good designs, and result in a unified framework for designing and assembling mechanisms.

APPENDIX

A Computing Mating Force Information

Consider figure 7e once again. We wish to make precise the notion that this is a force diode.

Force diodes are interesting and useful, in that we can use them to build objects that are easier to mate than to take apart. More generally, we wish to compute the forces required to assemble and disassemble our parts, because we must determine that excessive forces that could damage the parts are never exerted. Conversely, if an assembly we have designed can be taken apart with very small forces or by a wide range of motions, then the mechanical connection between the parts may be be insufficient. By extending our algorithm to calculate the forces the robot is required to exert (external forces) and the forces the parts experience while mating, we can develop a general tool that performs all these computations.

The forces and torques experienced by the root as result of the interaction of the pawls with the obstacles can be computed by considering the force balances on the pawls. We shall show the computation of the force and torque at the hinge point of a single pawl due to its contact. This information is useful to ascertain that the pawl does not experience excessive loading during assembly. It is straightforward to transform this force and torque into some other coordinate frame on the root body. The total force and torque on the root is found by summing the contributions of the individual pawls. The total force and torque information is important since it has to be supplied by the robot or assembly machine.

First of all, we observe that if the pawl sticks due to friction (as in the case of a friction diode) or due to incompatible kinematic constraints (as in the case of a kinematic diode), the forces will be infinite as time is increased. This is due to the assumption that the root and pawl are rigid bodies, but even in practice, we expect the forces to get extremely large. Hence, we need only to examine the case of the pawl sliding on the obstacles. Second, our results provide us with a mapping from time to configurations, i.e., to the angles of the pawl. Hence we shall derive the dependence of force on the configuration θ instead of on time.

We first consider the type-B contact. The torque τ at the hinge point is only due to the displacement of the pawl from its resting configuration, θ_0. Hence,

$$\tau = k(\theta - \theta_0), \tag{33}$$

where k is the torsional stiffness of the spring connecting the pawl to the root. Now the force on the contact point during quasi-static sliding is some multiple of the vector $\boldsymbol{f}_e$, i.e.,

$\boldsymbol{f} = \alpha \boldsymbol{f}_e$. ($\boldsymbol{f}_e$ is the vector along the friction cone edge in the oposite direction of sliding. Therefore, balancing the torques on the pawl about the hinge point,

$$\tau = \boldsymbol{r} \times \boldsymbol{f} = k(\theta - \theta_0). \tag{34}$$

Simplifying and rearranging terms, we can express α as a function of only θ:

$$\alpha = \frac{k(\theta - \theta_0)}{R_\theta \boldsymbol{p}_i \times (\boldsymbol{n}_j - \mu \boldsymbol{v})}. \tag{35}$$

The type-A case is similar. As in Section 7, we redefine $\boldsymbol{r}$ to be the vector from the hinge point to the contact vertex, $\boldsymbol{r} = \boldsymbol{p}_j - \boldsymbol{p}$, and $\boldsymbol{f}_e = R_\theta \boldsymbol{n}_i - \mu \boldsymbol{v}_\theta$. We have written the vector in the sliding direction as $\boldsymbol{v}_\theta$ instead of the usual $\boldsymbol{v}$ to emphasize that the direction depends on θ; more precisely, $\boldsymbol{v}_\theta = R_{\theta \pm \frac{\pi}{2}} \boldsymbol{n}_i$. Then performing a torque balance as in the type-B case, we get

$$\tau = \boldsymbol{r} \times -\boldsymbol{f} = k(\theta - \theta_0). \tag{36}$$

The negative sign appears in front of $\boldsymbol{f}$ since it was defined as acting on the obstacle. Simplifying and rearranging,

$$\alpha = \frac{k(\theta - \theta_0)}{(R_\theta \boldsymbol{n}_i - \mu \boldsymbol{v}_\theta) \times (\boldsymbol{p}_j - \boldsymbol{p})}. \tag{37}$$

The above expressions could be used directly to numerically compute the maximum force during sliding, using well known one-dimensional optimization methods. However, the maximum force and torque computation for a *single* pawl turns out to be much simpler. We define a *sliding segment* as a connected interval of time in which the pawl is sliding along an obstacle feature, without changing the contact topology. Since the torque τ varies linearly with θ, its maximum clearly occurs at a boundary of a sliding segment. We shall show below that the force maximum in a sliding segment can also occur only at a boundary of the segment. Hence one need only check the force and torque at these transition points. We have thus reduced the apparently continuous problem of finding the maxima to a discrete one.

Proposition A.1 *Let*

$$\alpha(\psi) = \frac{A\psi + B}{C \sin \psi}, \tag{38}$$

Subject to

$$\alpha(\psi) > 0 \ \text{ for all } \ \psi \in I = [\psi_{min}, \psi_{max}]. \tag{39}$$

Then α does not achieve a relative maximum (and hence a global maximum) in the interior of I.

Proof. We look at the relative extrema of α in I and show that they are minima. Differentiating with respect to ψ,

$$\alpha' = \frac{\sin \psi A - (A\psi + B)\cos \psi}{C \sin^2 \psi} \tag{40}$$

At a relative extremum, $\alpha' = 0$; hence the numerator of Equation 40 must be zero. Computing the second derivative of α at the extremum, and simplifying, we find

$$\alpha'' = \frac{A\psi + B}{C \sin\psi} = \alpha > 0. \tag{41}$$

Hence, the extremum is a relative minimum. □

Our claim follows by showing that our expressions for α are of the form required by the proposition. First, notice that for both Equations 35 and 37, the denominators simplify to the form $C_1 \sin\theta + C_2 \cos\theta$, where C_1 and C_2 are independent of θ, which can be easily rewritten as $C \sin(\theta + \phi)$, where $C = \sqrt{C_1^2 + C_2^2}$ and $\phi = \tan^{-1}\frac{C_2}{C_1}$. Hence, by making the substitution $\psi = \theta + \phi$, the expressions for α have the desired form for both type-B and type-A contacts. Finally, the requirement that $\alpha > 0$ in the sliding segment is nothing but the requirement that to maintain contact, the force has to be a positive multiple of $\boldsymbol{f}_e$ (see Section 6). Hence the above proposition applies, and the maximum α and hence the maximum force, can only occur at the boundaries of a sliding segment.

With this extension to our algorithm, our system can compute the force required in assembly and disassembly, and also the forces experienced by the pawls during execution. However, the computation of forces is not exact (because they involve transcendental functions that cannot be "rationalized" as we do in sec. 8 for pure kinematic constraints to make them algebraic); and hence this extension makes our algorithm approximate. (The use of provably good polynomial approximations to the transcendental functions will make the algorithm a provably good approximation algorithm, however). The computation of mating forces is perhaps one of the most important aspects of our algorithm for design for assembly.

B Incorporating Uncertainty in Control and Initial Conditions

B.1 Assumptions

Real robots are subject to significant uncertainty and error in sensing and control. For this reason we would like to generalize our algorithm to deal with uncertainty in initial conditions, and uncertainty in control. In this section we show how this may be done, using a simple model of uncertainty. We can easily extend the simple algorithm of Section 9 to handle a very simple form of uncertainty. We expect that the sophisticated sweepline algorithm could be similarly extended, but we have not done so.

The introduction of uncertainty changes the complexity of our naïve algorithm from $O(n^2 \log n)$ to $O(n^3 \log n)$; the algorithm remains combinatorially precise and algebraic.

First, we assume that the initial position $\boldsymbol{p}_0$ of the root body is known to start within some "start region" R. This region represents uncertainty in the initial conditions. We will still assume that the root body is controlled as a "moving constraint", parameterized by time. However, we assume that it is controlled by something like a generalized spring controller with feedback position correction. This model is very similar to that analyzed by Buckley (the spring-damper model) [Buc87], and we adopt it because Buckley's error analysis is convenient algorithmically. However, we will not use the damper component of the model to slide on surfaces. Under these assumptions, the set of possible positions that the root position $\boldsymbol{p}$ can reach (in free space, without obstacles) is bounded by a cylinder. In short, we incorporate sensing in control by assuming that with initial conditions $\boldsymbol{p}_o$ in R, the set of positions that the root position $\boldsymbol{p}$ can reach is contained in a cylinder $T_{\dot{\boldsymbol{p}}}$ starting at R. See fig. 23.

See eq. (2). The geometric formalization of our assumptions is that (i) $\boldsymbol{p}_o \in R$, and (ii) for all times t, the root position $\boldsymbol{p}(t)$ lies in a cylinder $T_{\dot{\boldsymbol{p}}}$ from R as in fig. 23. The practical realization of these assumptions requires (i) a bound on sensing errors so that the initial position can be ascertained to some known accuracy, and (ii) a feedback control system with bounded error, such as that of [Buc87].

B.2 Algorithms

B.2.1 Motion Prediction under Uncertainty is Computable

It is immediately clear that with this formulation of uncertainty, the motion prediction problem (to predict all possible outcomes from R subject to $T_{\dot{\boldsymbol{p}}}$) is decidable. We see this as follows. First, our algorithm for motion prediction from a single initial condition given $\dot{\boldsymbol{p}}$ (as described above) defines an algebraic predicate $F_{\dot{\boldsymbol{p}}}(\boldsymbol{p}_o, \mathrm{z})$ that decides whether a configuration z is reachable from initial condition $\boldsymbol{p}_o$ given commanded motion $\dot{\boldsymbol{p}}$. In effect, our algorithm computes the entire set

$$\exists \mathrm{z} \quad F_{\dot{\boldsymbol{p}}}(\boldsymbol{p}_o, \mathrm{z}). \tag{42}$$

That is, our algorithm outputs the entire set $Z(\boldsymbol{p}_o, \dot{\boldsymbol{p}})$ satisfying (42). Thus we can define another predicate

$$\mathcal{F}(\boldsymbol{x}, \mathrm{z}) \iff (\exists \mathrm{z})\ (\exists \boldsymbol{x} \in R) \quad F_{\dot{\boldsymbol{p}}}(\boldsymbol{x}, \mathrm{z}). \tag{43}$$

Correspondingly, predicates can be defined to decide all surfaces on which the pawls might stick, all surfaces the pawls might slide on, etc. Since the predicates are algebraic, they are decidable.

B.2.2 A Practical Algorithm

More practically, we can define a direct algorithm with a computational geometric flavor. Assume that generically, only one initial contact can occur at a time. Imagine that we "project" R onto the obstacle environment in "direction" $\dot{\boldsymbol{p}}$. This is similar to "intersecting" $T_{\dot{\boldsymbol{p}}}$ with the environment. That is, we sweep the flexible body along $\dot{\boldsymbol{p}}$ from R, and find all edges of the environment it can hit. We find all initial contact tuples

$$(g_A, t_{c_{min}}, t_{c_{max}}, g_B, \boldsymbol{x}(t_c))$$

such that pawl feature g_A can strike obstacle feature g_B at times t_c, $t_{c_{min}} \leq t_c \leq t_{c_{max}}$. Note that at initial contact of (g_A, g_B), the orientation of g_A and the other pawls is unchanged.

$\boldsymbol{x}(t_c)$ is a function that maps initial contact times to root positions. Hence, the initial contact tuple represents the fact that for $t_{c_{min}} \leq t_c \leq t_{c_{max}}$, pawl feature g_A can strike obstacle feature g_B if the root position is at $\boldsymbol{x}(t_c)$.

For example, in fig. 23, assume the pawl is a line segment anchored at $\boldsymbol{p}$ (as in fig. 12). There would be three initial contact tuples, one for each type (B) csurface of $\{ v \} \times \{ 1, 2, 3 \}$. We call the set of initial contact tuples the *projection* $\pi_{\dot{\boldsymbol{p}}}(R)$ of R under $T_{\dot{\boldsymbol{p}}}$.

Given the initial contact tuples, we chose a "sample point" for each one, and run our standard, no-uncertainty algorithm from that sample point. The algorithm is given below.

Algorithm for Motion Prediction Under Uncertainty

1. *Compute the initial contact tuples* $\pi_{\dot{\boldsymbol{p}}}(R)$.
2. *For each initial motion tuple* $(g_A, t_{c_{min}}, t_{c_{max}}, g_B, \boldsymbol{x}(t_c))$, *do*
3. *Assume type (B) contact, so that* g_B *is an obstacle edge.*[16] *Segment* g_B *into 3 regions:*
4. *The sliding region* e_1,
5. *The sticking region* e^*
6. *and the portion* e_0 *of* e *that cannot be hit initially.*
7. *Output* e^*.
8. *Choose a sample point* $\boldsymbol{x} \in e_1$.
9. *Compute and output the reachable set* $Z(\boldsymbol{x}, \dot{\boldsymbol{p}})$ *(eq. (42)) using our motion prediction algorithm for perfect initial conditions and no uncertainty.*

As usual, let n be the geometric complexity, that is, the product of the number of moving object features and the number of obstacle features. Then there are at most $O(n)$ initial contact tuples. Thus the loop is executed $O(n)$ times. e_1, e^*, and e_0 have size at most 2. Hence there are at most $O(n)$ sample points $\boldsymbol{x}$, and so the basic algorithm is called $O(n)$ times as a subroutine in the last step. Since the basic naïve algorithm runs in time $O(n^2 \log n)$, that means the algorithm for motion prediction under uncertainty takes time $O(n^3 \log n)$.

B.3 Extensions and Limitations

Ideally, one would like to extend our model of uncertainty to be more realistic. Ideally it should cover uncertainty in the direction of the moving constraint, and uncertainty in initial orientation. As of now, we hope it is rich enough to model interesting phenomena, but still be computationally feasible. Much work remains to be done in reasoning about assemblies in the presence of significant uncertainty. Our initial algorithm is just a start.

[16]The case for type (A) contact is very similar.

Acknowledgments: We would like to thank Randy Brost for very helpful comments on these ideas and on a draft of this paper, and to thank Mike Caine for helpful discussions. We are also very grateful to John Hopcroft for his enthusiasm and intellectual support of this work.

References

[ASS] Agarwal, A., M. Sharir and P. Shor. *Sharp upper and lower bounds for the length of general davenport-Schinzel sequences,* Tech. Rept. 332, Comp. Sci., Dept., Courant Institute (1987).

[BD] Briggs, A. and B. Donald. *Geometric algorithms for simulation of quasi-static mechanical systems under control uncertainty,* Unpublished MS, Cornell University. (1990).

[Boh92] K.-F. Böhringer. Computational Aspects of the Design of Micro-Mechanical Hinged Structures. AAAI'92 Fall Symposium on Design from Physical Principles, Cambridge, MA (October 1992)

[BR79] O. Bottema and B. Roth. *Theoretical Kinematics.* North Holland, Amsterdam, 1979.

[Bri] A. Briggs, Amy. (1989) *An Efficient Algorithm for One-Step Compliant Motion Planning with Uncertainty.* 5^{th} ACM Symposium on Computational Geometry, Saarbrucken, Germany. (To appear) in *Algorithmica*, **8**, (2), 1992.

[BLP] Brooks, R., and T. Lozano-Pérez. "A Subdivision Algorithm in Configuration Space for Findpath with Rotation." *Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, August, 1983.

[Bro] Brost, R.. *Computing Metric and Topological Properties of Confuguration Space Obstacles,* IEEE Int. Conference on Robotics and Automation, Scottsdale, AZ (1989).

[Bro2] Brost, R.. *Analysis and Planning of Planar Manipulation Tasks,* PhD. Thesis, CMU, CMU-CS-91-149 (1991).

[BM] Brost, R. and Mason, M.. *Graphical Analysis of Planar Rigid-Body Dynamics with Multiple Frictional Contacts,* Preprints, 5^{th} Int'l. Symp. on Robotics Research, Tokyo, Japan, August. pp 367-374 (1989).

[Buc87] Buckley, S. J.. *Planning and Teaching Compliant Motion Strategies,* Ph.D. Thesis. MIT, Department of Electrical Engineering and Computer Science, 1987. Also MIT-AI-TR-936 (1987).

[Can86] John Canny. Collision detection for moving polyhedra. *IEEE Trans. PAMI*, 8(2), March 1986.

[Can89] John Canny. On computability of fine motion plans. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 177–182, 1989.

[CR] Canny, J., and J. Reif. "New Lower Bound Techniques for Robot Motion Planning Problems", *IEEE FOCS* (1987).

[Don87] Donald, B. R.. *A Search Algorithm for Motion Planning with Six Degrees of Freedom,* Artificial Intelligence, 31 (**3**) (1987).

[Don88a] B. R. Donald. A geometric approach to error detection and recovery for robot motion planning with uncertainty. *Artificial Intelligence*, 37((1-3)):223–271, Dec. 1988.

[Don88b] Bruce R. Donald. The complexity of planar compliant motion planning under uncertainty. In *Proc. ACM Symposium on Computational Geometry*, June 1988. Revised version in *Algorithmica*, **5** (3), 1990 pp. 353-382.

[ELP] Erdmann, M. and T. Lozano-Pérez, T.. *On Multiple Moving Objects,* Algorithmica, (2): 477-521 (1987).

[Erd84] Michael A. Erdmann. On motion planning with uncertainty. Technical Report 810, MIT AI Laboratory, 1984.

[Erd86] Erdmann, M.. *Using Backprojections for Fine Motion Planning with Uncertainty,* IJRR Vol. 5 no. 1 (1986).

[Erd91] Erdmann, M. A.. *A Configuration Space Friction Cone.,* Proceedings of the 1991 IEEE Workshop on Intelligent Robots and Systems, Osaka, Japan, pp. 455–460. (1991).

[FHS] Friedman, J., J. Hershberger and J. Snoeyink. *Compliant Motion in a Simple Polygon,* 5^{th} ACM Symposium on Computational Geometry, Saarbrucken, Germany. (1989).

[GSS] Guibas, L., Sharir, M., and Sifrony, S.. *On the general motion planning problem with two degrees of freedom,* Proc. ACM Symp. Comp. Geom., Urbana, Il (1988).

[HS] Hart, S., and Sharir, M.. *Nonlinearity of Davenport Schinzel sequences and of generalized path compression,* Combinatorica, (2) pp. 209-233 (1987).

[KS] Kedem, K., and Sharir, S.. *Efficient algorithms for planning translational collision-free motion of a convex polygonal object in 2-dimensional space amidst polygonal obstacles,* Proc. ACM. Symp. Comp. Geom., pp 75-80 (1985).

[JA] L. Joskowicz and S. Addanki. "Innovative Shape Design: A Configuration Space Approach." Courant Institute of Mathematical Sciences, New York University, Technical Report 356, March 1988.

[LMT84] T. Lozano-Pérez, M. Mason, and R. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1), 1984.

[LP83] Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, February 1983.

[Mas82] Matthew T. Mason. *Manipulator Grasping and Pushing Operations.* PhD thesis, M. I. T., June 1982.

[Ma84] Mason, M.. *Personal Communication,* (1984).

[Pai88] Dinesh K. Pai. *Singularity, Uncertainty and Compliance of Robot Manipulators.* PhD thesis, Cornell University, Ithaca, NY, May 1988.

[PL92] D. K. Pai and M. C. Leu. Genericity and the singularities of robot manipulators. IEEE Transactions on Robotics and Automation, 1992.

[TM87] G. G. Trantina and M. D. Minnichelli. The effect of nonlinear material behaviour on snap-fit design. In *ANTEC '87*, pages 438–441, 1987.

[Whi76] Daniel E. Whitney. Force feedback control of manipulator fine motions. In *Joint Automatic Control Conference*, pages 687–693, West Lafayette, Indiana, 1976.

[Whi82] Daniel E. Whitney. Quasi-static assembly of compliantly supported rigid parts. *J. Dynamic Systems, Measurement, and Control*, 104:65–77, March 1982.

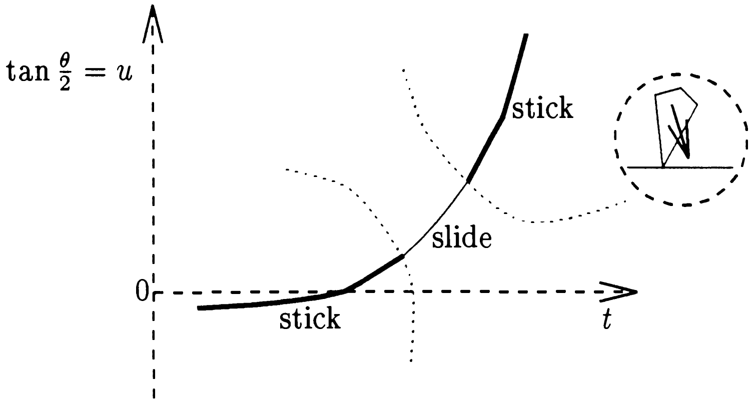


Figure 22: Sticking events and qualitative dynamical regions.

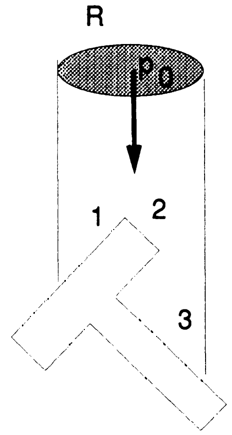


Figure 23: With uncertainty, the root position starts out in some region R. The control system ensures that as time evolves, the root position stays in a cylinder $T_{\dot{p}}$.