1986

Simulation of Physical Systems from Geometric Models

Christoph M. Hoffmann
Purdue University, cmh@cs.purdue.edu

John E. Hopfcroft

Report Number:
86-635

SIMULATION OF PHYSICAL SYSTEMS FROM GEOMETRIC MODELS

Christoph M. Hoffmann

John E. Hopcroft
Cornell University

CSD-TR-635
November 1986

Simulation of Physical Systems from Geometric Models[1]

Christoph M. Hoffmann

Department of Computer Science
Purdue University
West Lafayette, Indiana 47907

John E. Hopcroft

Department of Computer Science
Cornell University
Ithaca, New York 14853

Abstract

The design of an extensible system is discussed in which the behavior of physical objects is simulated from their models. Complex objects can be defined in a multiplicity of domains, including their geometric shape, their dynamic response to applied forces, and their controlled behavior. In response to unforeseen changes, e.g., for unexpected collisions, the object models are modified automatically during the simulation.

[1] Supported in part by National Science Foundation Grants ECS 83-12096, DCR 85-02568 and DCR 85-12443, and by ONR Grant N0014-86-K-0465.

1. Introduction

Model driven programming, where code for automatic assembly of objects is derived from a data base description of the assembly has long been a dream of researchers. A major stumbling block is automatic motion planning, presently a forbidding task requiring extensive mathematical knowledge and prodigious resources. A first step towards model driven programming that does not require a solution to the motion planning problem is a model driven simulation system. Such a system requires a collision detection algorithm but not a complete motion planning algorithm.

When considering how useful a proposed strategy for motion planning might be, a geometric simulation driven by an emulated program is required. Yet such an approach is limited whenever the dynamic aspects of the situation are important. Consider the grasping strategy for a hinge plate by Paul [14] in which use is made of both the situational geometry as well as the friction between the hinge plate and the work table. This strategy could not be verified by a system based purely on geometry. What is needed is a model-driven simulation system that duplicates the true behavior in the work cell as accurately as required by the nature of the problem. Such a system would need the capability to simulate motion of objects under external forces and it could be used to verify many other aspects of off-line robot programming as well.

In fact, such a simulation system, driven largely from a geometrical model, would have wide spread applicability: It could be used in electronic prototyping to verify aspects of a design such as the removability of each board in a computer frame for servicing or the proper unfolding of an antenna on a satellite. It could be used to simulate the workings of mechanisms, either for design refinement or for training. In fact, it could become the basis of a sophisticated tool for engineering design and analysis. A system of this kind would have substantial payoffs: It would allow design changes much later in the design cycle since revalidation only involves rerunning the validating algorithms. Furthermore, devices designed and developed to operate in unusual conditions not easily approximated in the laboratory such as the deployment of an antenna in space that will not support its weight under gravity can easily be prototyped.

Previous work on simulation is too extensive to discuss in detail. However, a number of unifying aspects and assumptions may be identified in the large majority of this work, including the following design limitations:

(1) Once a scenario to be simulated has been modeled, the simulation program cannot modify this model in significant ways during the course of the simulation. Yet such self-modification is needed to account for unforeseen changes in the scenario, e.g., when an unexpected collision takes place.

(2) The simulation system is specialized to a fixed spectrum of physical properties and behavior. It is not normally possible to extend the *physical* coverage of such systems.

In the more specific domain of robot simulation systems, the dynamic aspects of the simulated manipulator often are not included, e.g., in the LM system [9]. When dynamics is accounted for,

e.g., in [5, 17], the simulation concentrates on the manipulator itself and does not include deep interaction with the surrounding environment. Some systems, such as ADAMS [2], show an evolution towards the goals we outline here, but do not have the capacity to respond to unexpected events and do not make essential use of geometric information. In [6], Gilmore describes a two-dimensional dynamic simulation system for polygons that is also capable of self-modification. It contains a number of ingenious geometric techniques which unfortunately do not generalize to the three dimensional case.

We describe a general simulation system that is designed to be extensible and to be capable of self-modifying object models to account for unforeseen changes in the object configurations. In such a system it must be possible to define arbitrary collections of objects and to simulate their physical behavior in a multiplicity of domains, including the geometric, dynamic, and controlled aspects of behavior. The complexity of implementing the system dictated a global design in which it is possible to substitute a new version of an existing system component without affecting the rest of the system. Indeed, it is possible to construct component interfaces in such a way that any system component can be modified or replaced by a new version as long as the interface information can be computed. For example, we are able to replace the solid modeler in the geometric component with any solid modeling system as long as a number of basic operations are implemented by it. Moreover, since a precise simulation in multiple modeling domains uses extensive resources, the system should be instructed in a language that permits abstracting out some domains whenever this is appropriate. For example, a well controlled active system does not need a full dynamic simulation when only kinematic aspects of the situation are of interest.

In this paper, we describe the design and implementation of a first version in which only the basic geometric, dynamic, and control aspects of modeled objects are considered. Throughout, our design is illustrated by a number of examples, demonstrating the approach we have taken. The exposition is organized as follows: First, a global view of the systems structure is presented. Thereafter, the individual components and subsystems are discussed. Then, some of the features of the specification and simulation language are explained. The interface structure of the system is explained after each major subsystem has been described.

2. System Overview

The object of this work is the simulation and analysis of systems of physical objects. The user specifies the shape, material composition, and mechanical interrelationship of objects. From this description the system constructs a variety of models, sometimes guided by additional user specifications, where each model captures the object behavior in a specific physical domain.

At this time, we have restricted the scope to the behavior of rigid bodies that can be hinged and interrelated in a number of ways. Given the description of shape and material composition, a set of motion equations is formulated automatically that expresses the dynamic behavior of the

objects in Newtonian mechanics. Composition of rigid bodies is made by connecting them through mechanical hinges (including temporary contact) and/or by force relationships such as mass-less springs and dampers. These generalized hinges are selected from a standard set, e.g., revolute joint, ball and socket joint, etc. It is planned to augment the set by a definitional method for constructing new types of hinges from existing primitives. As primitive objects are combined through hinges, their motion equations are modified to correctly reflect the behavioral constraints so introduced.

Either with a single body, or else with a hinge, one may associate a control system. This system, which must be modeled explicitly, actuates components either by outright prescription of motion (perfect control), or else by applying forces and torques at selected features. These forces and torques can be given by a user-supplied program that may make use of the current state of the mechanical system.

Having defined a mechanical system, possibly with active components, a simulation is done that captures the behavior of the physical system according to the various models. Ordinarily, this is a cycle that begins with solving the motion equations accounting for hinges and for forces applied through control. Following the determination of new accelerations and constraint forces, the position of the individual bodies is updated and graphically rendered. In response to certain events, the models of the physical system may have to be changed. For example, it is possible that two objects formerly in contact separate, or that two objects collide. In each such event, an analysis is performed whose outcome is an appropriate modification of the state of the physical system, and of the models describing its now different future behavior.

Globally, the system is divided into three major components: A *definitional system* that permits definition of objects and their behavior, including the instantiation of objects and the declaration of world characteristics such as gravity; an *analysis system* that simulates how the defined objects behave over time and handles exceptional events; and a *report system* that gives summaries of the simulation by graphically rendering key scenes or numerically summarizing key aspects such as internal forces, accelerations, etc.

The major system components are subdivided into subsystems, each with a specific responsibility. For instance, the definitional component coordinates a number of modelers, each modeling different aspects of object characteristics and behavior. The analysis component directs subsystems concerned with solving differential equations, interpreting solutions of dynamic equations, interrogating the modeling subsystem for possible interference and collision, and updating models in response to exceptional events. Finally, the report system contains components capable of visually rendering key events or animating the simulation, and giving textual summaries of periodic or exceptional events.

The organization of the system is shown graphically in Figure 2.1. External program emulation is used for complex control systems, and is synchronized through a common clock. In the prototype presently implemented the program emulation has not yet been so separated. Rather,

user supplied subroutines for control are automatically incorporated at the interface level and are called at the appropriate times from the event handler.

A longer term goal is to extend the analysis system so that functions other than pure simulation can be supported. For instance, in order to determine the needed physical strength of a revolute joint, the constraint forces acting at the joint must be monitored and related to material properties of the hinge. We would also like to extend the physical coverage by adding finite element capabilities.

3. The Definitional System

Objects are either *primitive* or *composite*. A primitive object is a rigid body of specified shape. Composite objects consist of a number of primitive objects that are joined by generalized hinges.

All objects are modeled in a number of domains describing categories of behavior. In each domain one constructs a model of the object's relevant characteristics. For instance, in the geometric domain the shape is described, and in the dynamic domain the motion equations of the object are formulated. The set of models describing a primitive object is fairly straightforward. A complex object is described in each domain by combining the models of its primitive components, and by making certain modifications and additions to the component models. These changes are needed to account for the nature of the interaction among the primitive components.

Currently, we have the following domains: abstract, geometric, control, interference, and dynamic. The different domain-specific models are coordinated through a fixed interface design. For the overall system to be viable, it must be possible to extend the list of modeling domains as the system matures and is used in a broader range of applications. For example, one may wish to add an electrical characteristics domain to model VLSI components. Moreover, since the design and implementation of the system requires man years of effort, we attempt to make use of existing components such as separately developed solid modelers and display packages.

3.1. Abstract Model

The abstract model of a primitive object consists of a name and a property list containing material, density, color, etc. A composite object is represented by a graph whose vertices represent primitive objects and whose edges represent relations between these objects. Examples of relations are touching, rigidly connected, hinged in a particular way, and so on. Each edge has a list of properties as do vertices, and these properties may be interrogated by other subsystems.

Example 3.1:

Consider the anthropomorphic shape shown in the motion sequence in Figure 3.1. It consists of

several rigid bodies linked by revolute joints. For each rigid body, there is a separate abstract model recording properties such as density and color, as well as the fact that these bodies are components of a composite object. To the figure itself there corresponds an abstract model that is essentially a graph whose vertices are the rigid components and whose edges represent the joints. □

From the specified properties other characteristics are derived. For instance, using density and volumetric properties of the geometric model, mass and inertia of a primitive object are determined, and given the material we can supply simple friction models automatically. Other properties can be added as needed when enlarging the number of modeling domains. For instance, elasticity properties will be needed when modeling deformable bodies.

3.2. Geometric Model

The geometric modeling subsystem represents both the shape and the position/orientation of objects in 3-space. In the definition phase, the subsystem is used to construct the shape of primitive objects and to position primitive components with respect to each other or relative to a global coordinate frame. The usual operation during simulation is to move an object to a new position/orientation. In addition, the subsystem supports a number of capabilities that are needed for automated operations, including locating the centroid and computing the volume, computing the volumetric tensor of inertia, determining the coordinates of object features, and determining the features in which two primitive objects touch or intersect.

In order to facilitate coordination between geometric and dynamic models, we presently assume that primitive objects are homogeneous and that the local coordinate frames agree. Heterogeneous rigid bodies must be modeled as composite objects whose primitive components are rigidly attached to each other.

Composite objects are constructed from primitive ones by composition operations. These operations connect objects by hinged or by rigid connections. Objects are hinged with a *make_hinge* primitive whose arguments identify the type of hinge to be constructed and the features where the hinge attaches. In the example above, each member of the figure has been modeled as a cuboid with cylinders attached. The members are then linked by a pin hinge.

3.3. Interference Model

In the course of simulation we need to ascertain whether any objects interfere. This can be done using the geometric model, but as geometric coverage is extended this approach quickly becomes too slow. The purpose of the interference model is to allow quick noninterference tests and to reserve the expensive interference test based on the geometric model to critical instances when objects are in close proximity.

In the interference model, an approximate hierarchical model of objects is defined permitting us to test that two objects do not intersect. The approximations are constructed using a basic shape primitive, understood by the geometric modeling system, and a procedure for testing whether two primitive shapes intersect. Presently we use the cuboid as basic shape primitive.

The approximation levels are as follows: First, every primitive object is enclosed by a single cuboid. On the second level of approximation, the object is approximated as the union of several cuboids. On the last level the geometric objects must be intersected. More intermediate levels can be added if the need arises.

A composite object has moving parts that may interfere with each other, hence only the primitive components are approximated. For example, the moving parts of the anthropomorphic shape in Figure 4.1 have been designated as not interfering with each other. However, certain pairs of components may be known a priori not to interfere with each other, and this fact is indicated by connecting the corresponding cuboids by an edge. In consequence, the interference model of an object is a layered graph, each layer representing a level of abstraction. The graph vertices each are a union of shape primitives, and the graph edges represent the noninterference relationship. At present this model must be defined explicitly, but it should be possible to automate much of the modeling work from the geometric specification.

3.4. Dynamic Model

The dynamic modeling subsystem represents an object by a local coordinate frame, a set of state variables, and equation schemata that summarize the relations between the changes to state variables, time-independent properties, and external forces acting on the object. Some pertinent information is obtained initially by interrogating models of the object in other domains. For example, the mass is determined by obtaining the density from the abstract model and the volume from the geometric model. The model is set up largely without explicit user direction.

Unlike the geometric model, the dynamic model of primitive components undergoes substantial modifications as these components are combined into composite objects. For this reason, equation schemata are needed rather than a set of fixed equations. Given a set of applied forces, the schemata are used to construct the proper equations for the specific situation. Moreover, when objects are combined into a composite object, certain equations are modified to account for internal constraint forces that appear.

A primitive object that is a rigid solid has state variables r, p, $\dot{r}$, and ω, corresponding to position, orientation (in Euler parameters), linear and angular velocity. Note that these variables are vectors. There are two vectorial equation schemata of the form $m\ddot{r}=F$ and $J\dot{\omega}+\omega\times J\omega=T$, where m is the mass and J the inertia tensor. Moreover, we assume that all external forces and torques have been combined into a single resultant force F and torque T, applied at mass center.

When primitive objects are combined into composite objects, the sets of state variables are unioned together. The combining operation usually imposes constraints on the state variables associated with it. Since these constraints involve two primitive components, a composite dynamic model is represented as a graph whose vertices are the dynamic models of the primitive components of the object modified by the addition of constraint forces, and whose edges are the composition operations and their implied constraint equations.

There is a primitive composition operation from which most other composition operations can be derived. This primitive operation constrains a point on the second object to remain in contact with a surface on the first object, thus constraining one degree of freedom in the relative motion of the two objects. The point so constrained is called the *hinge* point. Implementing other composition operations as a suitable combination of this primitive has the advantage that the derived compositions are then expressed independent of modifications of the modeling systems.

When composing two objects by this primitive, we need to identify a surface of a primitive component of the first object and a point on a primitive component of the second object. The ensuing constraint equation linearly relates the relative accelerations of the respective primitive components. Moreover, the constraint force X transmitted at the hinge point is normal to the surface of contact. To account for it, the motion equation schemata of the two primitive components must be modified, where X by convention acts negatively on the first, and positively on the second component. So, when composing two rigid bodies, the following equations result:

$$m_1\ddot{r}_1=F_1-X$$
$$m_2\ddot{r}_2=F_2+X$$
$$J_1\dot{\omega}_1+\omega_1\times J\omega_1=T_1-c_1\times X$$
$$J_2\dot{\omega}_2+\omega_2\times J\omega_2=T_2+c_2\times X$$

In these schemata, m_i is the mass of body i, J_i its inertia tensor, F_i is the resultant external force and T_i the resultant external torque on body i. Furthermore, c_i is the vector from mass center to the hinge point, for body i.

Given external forces and torques, the two hinged bodies account for 12 scalar equations with 15 unknowns. Three scalar equations are added for the primitive hinge. Assuming there is no dry friction at the point of contact, two of these equations are $s_1\cdot X=0$ and $s_2\cdot X=0$, where the s_i are two linearly independent vectors parallel to the plane of contact. This says that the constraint force must act normal to the plane of contact. The third equation expresses that the point remains in contact with the plane. It is the derivative of the equation $n\cdot v_p=0$, where n is the normal to the plane of contact and v_p is the velocity of p relative to the plane of contact. This is an unfamiliar formulation that has the advantage of simplifying the evaluation of the motion equations at each time step.

If the hinge is due to temporary contact of two bodies with, say, a vertex of body 2 touching a surface of body 1, then the resulting hinge cannot sustain tension. In consequence, whenever X

becomes 0 or negative the hinge "breaks" and both the force X, as well as the three constraint equations must be deleted. Accordingly, there is a *break_hinge* operation that changes the equations of the dynamic model. This is done by deleting the constraint equations associated with the graph edge that represents the hinge and by deleting the corresponding constraint force terms in the motion equations of the two primitive objects no longer connected by the hinge.

Recall that we have formulated the primitive hinge for the friction-less case. If dry friction occurs, for example when modeling temporary contact between two objects, the text book approach is to formulate two different sets of equations, one for the case when slippage occurs, the other for the case in which the friction is sufficient to prevent relative movement between the objects in contact. For systems of many contacting bodies this leads to an exponential growth of the number of alternative models that need to be investigated. In Section 4 we discussed our way of avoiding this problem.

Example 3.2:

We consider the hinged linkage of Figure 3.2 in two dimensions. With rod dimensions of 2 and 8, we develop the motion equations needed to describe the composite object dynamically. There is an inertial world reference frame with x,y coordinates. Each rod has a local coordinate frame that at time t is in position r_i and tilted by the angle θ_i with respect to the x-axis; see also Figure 3.3. Moreover, the hinge point of the link with the active system is at position r_0: $(x_0=b\sin(\omega t), y_0=0)$. Assuming no external forces are acting, i.e., all movement is induced by the constrained motion of the active system, the following equations describe the behavior of the rods:

$$m_1\ddot{r}_1=X_{10}-X_{12}$$
$$m_2\ddot{r}_2=X_{12}$$
$$J_1\ddot{\theta}_1=c_{10}\times X_{10}-c_{12}\times X_{12}$$
$$J_2\ddot{\theta}_2=c_{21}\times X_{12}$$

Here m_i is the mass of body i and J_i is the moment of inertia, about the z-axis. The vectors c_{ij} are vectors from the origin of the local coordinate system i to the hinge point connecting body i with body j. For instance, $c_{12}=(4\sin\theta_1+\cos\theta_1, \sin\theta_1-4\cos\theta_1)$. The kinematic constraint equations, added to the dynamic model when the rods were connected by the pin hinges, are

$$\ddot{r}_1-\dot{\theta}_1^2c_{12}+\ddot{\theta}_1\hat{c}_{12}=\ddot{r}_2-\dot{\theta}_2^2c_{21}+\ddot{\theta}_2\hat{c}_{21}$$
$$\ddot{r}_0-\dot{\theta}_0^2c_{01}+\ddot{\theta}_0\hat{c}_{01}=\ddot{r}_1-\dot{\theta}_1^2c_{10}+\ddot{\theta}_1\hat{c}_{10}$$

where $\hat{c}$ is the vector c rotated by $\pi/2$. That is, if $c=(u,v)$ then $\hat{c}=(-v,u)$. Note that the vectors c_{ij} are determined from the geometric model when the hinges were defined. The second constraint equation simplifies since $c_{01}=(0,0)$. Moreover, since $r_0=(b\sin(\omega t),0)$, we have $\ddot{r}_0=(-b\,\omega^2\sin(\omega t),0)$. This equation is obtained from the control model of the active system. □

3.5. Control Model

Many situations we wish to model involve program driven objects. For example, we may wish to debug off-line a robot program implementing an approach strategy for the gripper to grasp an object. The robot arm will have actuators that apply forces and torques at the joints. The actuators receive signals from a control system that processes two kinds of signals. The first type includes sensing signals that must be supplied from the simulation of the world model. The second kind of signal originates from a program driving the control system. Sensing signals are generated by allowing state variables to be read, dynamic forces to be determined, and geometric distances to be computed.

In the simplest situation, the acceleration of an object is controlled directly. This is the case in Example 3.2: The active system is constrained to move as the function $(b\sin(\omega t),0,0)$, i.e., the acceleration vector can be defined as $(-b\,\omega^2\sin(\omega t),0,0)$.

Example 3.3:

In [12], Chapter 7, an actuator is modeled. Ignoring Coulomb friction but accounting for both the actuator gain and viscous damping, the equation $\dot{\omega}=K_1u-K_2\omega$ models the actuator, where u is the input signal, K_2 the damping factor, and ω and $\dot{\omega}$ angular velocity and acceleration at the actuated joint. K_1 depends on the effective inertia of the actuated link and on the actuator gain, and could be considered constant in simple models. More complicated control models establish a relationship between K_i and other state variables, thereby incorporating feedback and feed-forward loops. □

In the linkage example acceleration was controlled by equating it with a specific function. In general, the control function can be supplied by a subroutine written in Lisp. In order to avoid problems of data dependency, these subroutines are structured as follows: As input, the value of any state variable or force from a prior time interval may be used. The output values determined are then available for reference at the current time.

Example 3.4:

In Section 4, a simulation sequence is shown in which an anthropomorphic figure rises from a sitting position. The motion is controlled by subroutines that apply certain torques at the hip, knee and ankle joints. The subroutines establish correct torque values by sensing velocities and position of the joints, and calculating the torques accordingly. They are user-supplied control models that are associated with each of the joints. The relevant state variable values are obtained by asking the interface procedures for the objects associated with the hinge and their current state.

While the subroutines in the motion sequence shown in Section 4 are not very general, more sophisticated procedures can be developed that contain control sequences to be executed in response to specific, interactively issued commands, thereby providing the infrastructure needed for a high-level manipulation language. Moreover, the torque values should be determined based not only on sensing the current state but also on the characteristics of the joint actuators they

model. □

3.6. Module Interfacing and Implementation

The system is implemented in Symbolics Common Lisp. The user communicates with the modeling subsystem through a user interface language that is described in Section 6. The definitions he makes are mapped to a sequence of internal instructions that engage various domain-specific modelers and create coordinating data structures. At certain places, for example in the dynamic modeler, other internal instructions are interpolated, thereby implementing automatic capabilities.

The set of internal instructions and the design of the coordinating data structures constitute the conceptual system implementation. They are fixed and mirror the steps in creating and operating models as outlined above. Each modeling subsystem is packaged by an interface that understands the internal instructions and data structures. At the interface level, internal instructions are executed by issuing modeler specific commands and formatting the data to be communicated. For example, the data structure formulated for a primitive object is as follows:

(1) It has a name, the type *primitive*, and a reference to the composite object(s) containing it directly.

(2) It is described as a list of two-element lists where the first element identifies the modeling domain, and the second is a model description some of whose details may be specific to the particular modeler and its implementation.

Similarly, a composite object has a name, type, and reference to the directly containing object. In place of a list of models, there is a composition list identifying the components and hinges of the object, and how they are linked.

To some depth, the domain-specific data structure is prescribed. For example, the geometric model consists of a transformation, a list specifying named features, and a shape description that is modeler-specific. While the format of the transformation is fixed, the format in which the shape description is given depends on the modeler. In our case it is a boundary representation, but it could change when the geometric modeler is altered. Since the overall structure of the interface is fixed, such changes are localized.

As example of an interface instruction, consider the mass computation of a primitive object. It involves obtaining the volume of the object with the *geometric_model_get_volume* instruction, directed to the geometric modeling interface, and the density with the *abstract_model_get_property* instruction, with argument *density*. The density is initially posted with the *abstract_model_set_property* instruction.

The flexibility and modifiability of the system is the result of strictly separating the abstract implementation, at the interface level, from the underlying concrete implementation. When large

data volumes are communicated between components on the abstract level, however, a price may have to be paid for uniform data formats. For example, both the geometric modeler as well as the rendering system need the shape description of objects, a very large and intricate data structure. It is advantageous if both routines can work from an identical, native structure, and in our implementation they do, rather than converting between different representations. This convention raises the difficulty of modifying these routines, since changes to the data format must be coordinated. Therefore, the device of sharing implementation-specific data must be carefully limited. In our system it is only used for the geometric shape description.

4. The Analysis System

Carrying out various engineering analyses on collections of objects requires a number of analysis and simulation packages. These packages make up the analysis system. At present we are primarily concerned with a program-driven dynamic simulation of a complex scene. The major components are an integration package that integrates the differential equations that arise in the dynamic simulation, an event handling routine that analyzes the solutions to the equations and upon detecting exceptional events alters the models defining the scene or updates the world state, and a language component that, by simulating a program instructing active agents in the scene, presents external stimuli to the simulation.

As in the object modeling system other components can be added. For example, a significant addition would be to model deformation, i.e., to incorporate finite element techniques.

4.1. Simulation Package

The simulation package integrates numerically the system of differential equations modeling object behavior in the various domains, for one time step. These equations are determined as follows: Using prior values, all programmed functions of the control models are evaluated. Substituting the resulting values, all (scalarized) dynamic equations and remaining control model equations are obtained by the event handler and are presented to the simulation system. The equations are in the form

$$Ax=b$$

where x is a vector of accelerations and constraint forces. The scalar entries of the matrix A and the vector b are functions of state variables, external stimuli, and time.

Exceptional events result in changes to this equation system, including the introduction or deletion of unknowns. When such events are infrequent, it is desirable to precondition this system so as to speed up its solution at each time step. Suitable methods have been proposed in, e.g., [4], and include triangularizing A in conjunction with solving the system through back

substitution.

It is possible that the system of equations is singular. In this case, additional assumptions must be made. The event handler is responsible for dealing with this situation. Once x has been determined at time t, all state variables can be updated accordingly. Before doing so, however, the event handler has to examine x and determine whether an exceptional event has occurred.

4.2. Event Handler

The event handler examines the state variables after each integration step in order to determine if an event has occurred that requires modification of the system equations. We are presently recognizing the following exceptional events:

(1) With updated state variables, the geometric model indicates an interpenetration of two objects. The moment of first contact is determined by interpolation or repeated subdivision and integration. Then an impact is modeled, and from the impact model all velocities are updated. If the contact persists, e.g., due to inelastic impact or as the result of friction, the two contacting primitive bodies are connected by a pressure-only hinge. This entails updating the models and building a composite object.

(2) The reactive force maintaining a physical contact between two objects has become negative, so the contacting bodies separate at that point. Here we must edit the model by removing the graph edge representing the lost contact, and deleting the constraint force from the motion equations of the two adjacent components.

(3) The system $Ax=b$ is linearly dependent. This typically happens when a rigid body is in contact at more than six points and the contact forces are unknown. In this situation a more sophisticated model of contact is needed. For example, we could model infinitesimal interpenetrations and corresponding restoring forces. More complicated approaches could model elastic deformations in greater detail.

(4) The system $Ax=b$ has no solution. Typically, a controlled variable cannot be satisfied. For example, we may prescribe the motion of two rods in a way that requires the two rods to interpenetrate. In this case the model is deficient, and the simulation cannot continue.

After all exceptional events have been accounted for, a new system of equations will result and is presented to the simulation system. If no new system results, or if no exceptional event ensued to begin with, then the state of the world is updated using the vector x, and the cycle repeats.

Example 4.1:

Consider the simulation sequence shown in Figure 4.1, where an anthropomorphic figure rises from a sitting position. A replay of the actual simulation is shown in which certain snapshots of the animation sequence are placed side-by-side. The dialogue in the left window shows the interaction with the animation replay tool.

Throughout the simulation of this motion sequence, the following cycle is iterated: The control models request current state variable values and present new torque values. Based on the current state variable values and torques, as well as gravity, the motion equations are collected and its terms are evaluated yielding a system of linear equations. This system is solved, and the resulting accelerations and constraint forces are examined. During the first step the hinge modeling the contact between the figure and the block breaks under tension. This entails a reformulation of the model. The hinge between feet and ground does not break as the pressure is maintained. A new update of the state variable values is proposed and tested for interference. None is detected, so all state variables are updated and the cycle repeats. □

Example 4.2: In Figure 4.2 an instance of a simulated linkage motion is shown. Three links, numbered 2, 3, and 4 from top to bottom, are connected in a chain to a fictitious link 1. To link 1, a perfect control system is attached that prescribes a sinusoidal motion along a line parallel to the ground. For clarity, link 1 is displayed as a small box. The left window shows the scalarized equations of motion collected for this specific time instance. For readability, the variables are named as follows: xi, yi, and zi refer to the linear accelerations of link i; wx$i$, wy$i$, and wz$i$ refer to the angular accelerations; finally, fx$i$, etc., refer to the constraint forces at the pin hinge between links i and i+1. The determined values are also displayed. □

4.3. Impact

The impact of two bodies is modeled in the usual way as an infinitesimal event that leads to instantaneous velocity changes without changes of position. Following [18], the equation schemata governing the response to impulsive forces and torques are

$$m\Delta\dot{r}=\hat{F}$$
$$J\Delta\omega=\hat{T}$$

where $\hat{F}$ is the impulsive force applied to the body and $\hat{T}$ is an impulsive torque applied to the body. $\Delta\omega$ is the change in angular velocity and $\Delta\dot{r}$ the change in linear velocity at mass center.

On collision without friction, there is an impulsive force acting normal to the common tangent plane of the point of collision, of unknown magnitude. It can be determined by the equation

$$v^A = -ev^B$$

relating the relative approach velocity v^B of the colliding points just prior to impact to the relative separation velocity v^A immediately after the impact, in the direction normal to the common tangent plane. Here e is the coefficient of restitution that depends on the material of the two colliding bodies, the geometry of the impacting features, and the approach velocity. The coefficient is in the range $0 \le e \le 1$. As first approximation, e may be taken as a constant derived from the materials of the colliding bodies.

When the colliding bodies are composite, the hinges transmit impulsive constraint forces and torques that are determined with the help of constraint equations that are analogous to the kinematic constraint equations for nonimpulsive forces and torques [18]. The situation is completely analogous to the determination of constraint forces in the motion equations in conjunction with the kinematic constraint equations. Note that nonimpulsive forces do not influence the behavior since the impulsive forces are very large by comparison. So the impact may be simulated using an analogous but separate equation system.

Example 4.2:

The impact model for the two rod linkage is derived as follows. Assuming that the two rods collide at u (Figure 4.3), we develop the equations for the instantaneous velocity changes. Again, the problem is considered in two dimensions. Let $\dot{r}_i^B$ be the velocity of rod i just prior to collision at mass center, $\dot{\theta}_i^B$ the angular velocity. Then the velocity of the colliding points on the two bodies prior to collision is given by

$$v_1^B=\dot{r}_1^B+\dot{\theta}_1^B\times d_{12}$$
$$v_2^B=\dot{r}_2^B+\dot{\theta}_2^B\times d_{21}$$

Similarly, with the superscript A indicating the time just after collision, we obtain

$$v_1^A=\dot{r}_1^A+\dot{\theta}_1^A\times d_{12}$$
$$v_2^A=\dot{r}_2^A+\dot{\theta}_2^A\times d_{21}$$

The rods are subject to an impulse $\hat{F}$ at the colliding points, and a reactive impulse $\hat{X}_2$ at the hinge between the two rods. In addition, there is a constraint impulse $\hat{X}_1$ acting at the hinge between the active system and rod 1. Consequently, we have the following motion equations:

$$m_1(\dot{r}_1^A-\dot{r}_1^B)=-\hat{F}+\hat{X}_1-\hat{X}_2$$
$$m_2(\dot{r}_2^A-\dot{r}_2^B)=\hat{F}+\hat{X}_2$$
$$J_1(\dot{\theta}_1^A-\dot{\theta}_1^B)=-d_{12}\times\hat{F}+c_{10}\times\hat{X}_1-c_{12}\times\hat{X}_2$$
$$J_2(\dot{\theta}_2^A-\dot{\theta}_2^B)=d_{21}\times\hat{F}+c_{21}\times\hat{X}_2$$

Now the impulse $\hat{F}$ acts normal to the line brought into contact by the collision. Hence

$$\hat{F}\cdot t=0$$

where t is parallel to the contact line. Moreover, approach and separation velocities of the colliding point are related as

$$(v_1^A-v_2^A)\cdot n=-e(v_1^B-v_2^B)\cdot n$$

where n is normal to the line of contact. In addition, the hinges imply the constraints

$$\dot{r}_1^A+c_{10}\times\dot{\theta}_1^A=\dot{r}_0$$
$$\dot{r}_2^A+c_{21}\times\dot{\theta}_2^A=\dot{r}_1^A+c_{12}\times\dot{\theta}_1^A.$$

The unknown quantities v_i^A, $\dot{r}_i^A$, $\dot{\theta}_i^A$, $\hat{F}$, and $\hat{X}_i$ can now be determined from these equations. Note that $\dot{r}_0=(b\,\omega\cos(\omega t),0,0)$, since we assume that the active system is perfectly controlled. □

4.4. Restoring Forces

When a rigid body is statically supported at more than six points during simulation, then the system $Ax=b$ will be indeterminate. Should these support points correspond to permanent linkages, the model must be reformulated. However, when bodies come into multiple contact, e.g., as shown in Figure 4.4, the problem cannot be avoided. Here a block A rests on three equal blocks B_i. Assuming all bodies are perfectly rigid, it is not possible to determine the exact load carried by each supporting block B_i.

A possible way of handling this situation is to permit infinitesimal interpenetration and restoring forces. In effect, one now conceptualizes the area of contact as elastic with restoring forces proportional to the interpenetration. Modifying the approach of [3] slightly, we proceed as follows. Assume given all forces acting on the body at time t. Compute the resulting acceleration for time $t+\Delta t$ while keeping the contact forces unchanged, thereby determining a velocity. For each contact point, compute the relative speed of approach v and consider the component v_n normal to the plane of contact. Then the contact force increments must be proportional to the relative interpenetration, i.e., we add as many equations of the form

$$\Delta F_i = k d_i ,$$

where k is the normal contact stiffness and d_i is the interpenetration depth at the i^{th} contact point. In a more sophisticated model shearing force resistance is also accounted for. Experimentation is needed to assess in what situations the approach is realistic and to explore alternatives.

4.5. Friction

Dry friction between contacting bodies is difficult to model analytically. The simplest model assumes a frictional force of magnitude μN, where μ is a constant depending on the material and the surface characteristics, and N is the magnitude of the normal force at the contact point. The direction of the frictional force is always opposite to the resulting motion. A typical difficulty arises from the fact that the precise distribution of the normal force over the area of contact is unknown. If sliding occurs, then the resulting motion may critically depend on the distribution [15, 16]. At this time, we assume a uniform distribution of the normal forces over the contact area. This is similar to the approach taken in [3].

The text book approach to formulating motion equations in the presence of dry friction is to formulate for each frictional contact two alternative sets of equations, one in which slippage does not occur, and the other one for slippage. For systems with many contacts this is not an acceptable approach. Instead, we deal with friction as follows: Each contact with dry friction is formulated as friction-less in the motion equations. Then it is determined what force is needed to counteract the resulting relative motion at the joint. If this force does not exceed μN in magnitude, it is added as external force acting at the joint, otherwise a force of μN is applied. The frictional forces so determined for time t are applied to the system at time $t+\Delta t$.

A more precise treatment of friction is possible if the frictional force is determined by iteration. As initial step, the frictional forces at time t are applied to determine the system solution for time $t+\Delta t$. Next, the normal forces are recalculated and used to refine the estimate of the friction forces assumed initially, and the calculation is repeated. We have presently no experimental data comparing the two methods.

4.6. Interface Considerations

The key aspect to the analysis package and its extensibility is the conception of the event handler. Basically, the event handler implements the major simulation steps in interface level operations. In consequence, it is in no way dependent on the implementation of the modeling subsystem. Ideally, the event handler should be programmable by the user in a high level analysis language. At this time, such a language does not exist and the user makes only minimal choices directly affecting the simulation.

The bulk of interaction between the analysis and modeling systems is provided by a routine for collecting equations, a routine for checking that the state variable increments and constraint forces make sense, and a routine for updating state variable values. These routines deal primarily with vectorial quantities and therefore work with implementation independent data structures. In conjunction with using the interface operations to formulate the procedures, independence from the modeling system's implementation is easy to achieve. For example, the routine for collecting equations begins its work with a list of all existing objects and hinges. By addressing first the control models of each object, and then the dynamic models, motion equations are collected whose terms are evaluated based on current information. Note that the control models use inputs based on the previous simulation cycle. The collection process results in the formulation of the system $Ax=b$.

State variable updates do not make sense if in the new position objects interpenetrate. Essentially a tentative update of state variables is performed and the resulting configuration is inspected. If no interference is found, the update becomes permanent. If contact forces vanish or become negative, the event handler is informed of the respective hinges and applies to each a break-hinge operation after the current cycle completes. See also [6] for a similar strategy.

5. Report System

The report system generates output showing how the simulation progresses and summarizing certain aspects. In the current version of the system, an animation is generated by displaying the updated geometric model of the world at regular intervals. It is possible to store the animation sequence and replay it. Ad-hoc tools also exist for instrumenting various parts of the system, thus we may monitor selected variables, display motion equations at certain instances, and the like. Eventually the report system must be enhanced by formulating systematic interfaces for

summarizing key events or for monitoring certain state variables. For example, one might wish to tabulate the impulses upon each impact, or might wish to log the constraint forces acting on a given body.

6. Definition Language

A scenario to be simulated is called a *world*. A world is defined by describing the *objects* in it and by placing these objects into an *initial configuration*. Also described are certain global properties such as the presence or absence of gravity. All detailed descriptions of objects, their shape, structure, etc., are really descriptions of generic objects, called *types*. When an object with such characteristics is wanted, one declares an instance of this type as part of the world description.

The user describes object (types) in a source language implemented by translation to interface operations. In describing primitive objects, the domain specific aspects are specified in subsections. Each subsection is a single expression or list, or, for more complex definitions, is enclosed in a **begin** ... **end** bracket. For instance, the rods of Example 4.2 were described by

```
primitive rod begin
   properties: (density: 2.0, color: red);
   geometry: cuboid(1, 3, 1)
            where begin
               top_hingepoint: (0, 1.6, 0);
               bottom_hingepoint: (0, -1.6, 0)
            end;
   dynamics: ;
end
```

Here the abstract properties density and color are identified, the geometric shape is defined as a cuboid with the appropriate dimensions, and certain features of this shape are named. These features can be referenced by the naming convention of [7]. Briefly, feature x of object a is referred to as $a.x$, and the names of features are inherited. E.g., if a is a component of composite object b, then feature x in a is referred to as $b.a.x$. The dynamic model is constructed automatically, so no specification is needed, but its inclusion must be indicated by stating the keyword *dynamics* followed by an empty description.

The active system controlling the movement of the upper rod is defined as

```
primitive driver begin
   geometry: none
            where begin
               hingepoint: (0,0,0)
```

```
            end;
        control: acceleration = acceleration_for_linkage_driver
    end
```

The driver has no material properties, yet a rudimentary abstract model is constructed to achieve a coordination of the other modeled aspects of the system. For hinging the active system with the upper rod, a special geometric description is needed in which the keyword *none* indicates that no associated geometric model exists. However, a local coordinate frame is needed so that the features by which the system will be hinged to the rod may be specified. This feature is referred to in the usual manner as "driver.hingepoint". No dynamic model exists, so any motion is governed by the control model. In the control model, we specify that accelerations are to be determined by a subroutine named "acceleration_for_linkage_driver".

For linking three rods and the active system together, we create a composite object as follows:

```
composite linkage begin
  components
    rod1: driver;
    rod2, rod3, rod4: rod;
  structure begin
    join rod2 to rod1 with ball.and.socket matching
      (top_hingepoint hingepoint)
    join rod3 to rod2 with ball.and.socket matching
      (top_hingepoint bottom_hingepoint)
    join rod4 to rod3 with ball.and.socket matching
      (top_hingepoint bottom_hingepoint)
  end
end
```

The components of the composite object are given names rod1 through rod4. They are primitive objects but could be composite in turn. The interconnection structure is given in the **structure** section and specifies that the rods are to be connected by ball and socket hinges. For each hinge the components to be connected are identified along with the feature in each that is to be the hinge point. For different joints more complex mating may be required. For instance, a pin joint requires mating two points on the axis of revolution. This implies a set of (linear) constraint equations that must be solved when instantiating the linkage in a world.

Before the simulation can begin, we must describe a world and declare all objects that are to be instantiated in it. The following description gives this information:

```
world example(x, y, z) begin
  components
```

```
    links: linkage;
structure
    place links by
            links.driver.center at (x, y, z)
        velocity = ((3.14, 0, 0) (0, 0, 0))
    properties: (gravity);
end
```

Here, the only object is of type linkage. Its placement is at the unspecified world location (x,y,z). The respective coordinate values are substituted by the user when invoking the simulation. The composite object links is given an initial linear and angular velocity.

A world simulation is then initiated by issuing the following instructions to the analysis and report subsystems:

```
simulate example(2,1,0) begin
    time: (0.05s, 60s);
    report: animation;
end
```

where a simulation of the defined world is requested for a period of 60 seconds with time steps of 0.05 seconds. The parametric start position for links.rod1 is chosen to be (2,1,0), and the simulation should be shown as an animation.

7. Discussion

We have described an experimental simulation system in which the geometry, dynamic and controlled behavior of models of interacting physical objects is simulated. The major design features include system modularity and extensibility, a flexible and friendly user interface, and the capacity to self-modify the models in response to exceptional situations, such as unanticipated collisions. The system is intended for experimentation in several different areas. It is used for developing user interfaces that will simplify the construction of simulations, and, eventually, for experimenting with different task strategies for gripping and rotating objects. Work also continues to refine the implementation and to incorporate more sophisticated techniques.

For small worlds, including all examples discussed here, the majority of time is spent collecting the motion equations and scalarizing them. To a large part this is so because presently no attempt is made to exploit model continuity over prolonged time spans, and much room for improvement exists. For larger worlds, the solution of $Ax=b$ is the dominant step. This can be alleviated by taking advantage of the sparseness of A.

To understand how the thrust of our system differs from other mechanical simulation systems, we review some of the characteristics of ADAMS [2], a very successful simulation and

analysis package. In ADAMS, a mechanism is defined as a system of interconnected rigid bodies, called *links*, that are connected by hinges of various kinds, such as ball and socket hinges, revolute joints, etc. The shape of links can be described by wire frame models in a user interface, but for the internal simulation the bodies are understood as local coordinate frames with mass and inertial properties. Note that mass and moments of inertia must be provided by the user. With each link one may associate *markers*, that is, distinguished points akin to named features in our geometric models. Markers are named and referred to by numbers. Two bodies are hinged by creating a standard joint between two markers. To orient the axis of revolution in a pin hinge, say, each marker must have a local coordinate frame with a suitable orientation relative to the associated link.

With joints one may associate *generators* that model actuators exerting, for example, constant torque. These generators may be supplied as subroutines. In addition, external forces may be applied at specified markers, as well as resisting or attracting forces between marker pairs. The latter may be used to effect an impact model: Associate with a pair of markers a distance dependent repelling force that is negligible except at very small distances.

It follows, that the system of motion equations never changes structurally in the course of the ADAMS simulation [2]. In particular, the user must anticipate all possible pairs of colliding points over the course of the simulation and declare them as markers. In fact, the self-modification of the models simulated is one of the basic capabilities in our system not found in any other mechanical simulation system [6]. When using the system as a tool to test motion planning strategies, for example, this capability is clearly needed.

8. References

[1] A. Ambler (1984)
"Robotics and Solid Modeling: A Discussion of Requirements Robotic Applications put on Solid Modeling Systems". *Robotics Research*, 2nd Intl. Symp., Kyoto, 1984, MIT Press, 1985, 361–367.

[2] M. Chace (1985)
"Modeling of Dynamic Mechanical Systems" CAD/CAM Robotics and Automation Institute and Intl. Conf., Tucson, Feb. 1985.

[3] P. Cundall, R. Hart (1985)
"Development of Generalized 2-D and 3-D Distinct Element Programs for Modeling Jointed Rock", Misc. Paper SL-85-1, US Army Corps of Engineers, Waterways Experiment Station, Vicksburg, Miss., Jan. 1985.

[4] S. Dubowsky, J. L. Grant (1975)
"Application of Symbolic Manipulation to Time Domain Analysis of Nonlinear Dynamic Systems", *J. of Dyn. Syst., Measurement and Control*, March 1975, 60–68.

[5] S. Dubowsky, R. Kornbluh (1984)
"On the Development of High Performance Adaptive Control Algorithms for Robotics", *Robotics Research*, 2nd Intl. Symp., Kyoto, 1984, MIT Press, 1985, 119–126.

[6] B. Gilmore (1986)
"The Simulation of Mechanical Systems With a Changing Topology," Ph.D. Dissertation, Mechan. Engr., Purdue University, August 1986.

[7] C. Hoffmann, J. Hopcroft (1985)
"Automatic Surface Generation in Computer Aided Design," *The Visual Computer 1*, Springer 1985, 92-100.

[8] J. D. Hollan, E. L. Hutchins, L. M. Weitzman (1984)
"STEAMER: An Interactive Inspectable Simulation-Based Training System", *AI Magazine* 5, 15–28.

[9] J. Latombe, C. Laugier, J. Lefebvre, E. Mazer, J. Miribel (1984)
"The LM Robot Programming System", *Robotics Research*, 2nd Intl. Symp., Kyoto, 1984, MIT Press, 1985, 377–391.

[10] T. Lozano-Perez, M. Mason, R. Taylor (1984)
"Automatic Synthesis of Fine-Motion Strategies for Robots", *Intl. J. of Robotics Research* 3:1, 3–24.

[11] M. Mason (1984)
"Mechanics of Pushing", *Robotics Research*, 2nd Intl. Symp., Kyoto, 1984, MIT Press, 1985, 421–428.

[12] R. Paul (1981)
"Robot Manipulators: Mathematics, Programming, and Control", MIT Press, 1981.

[13] R. Paul, H. Zhang (1984)
"Robot Motion Trajectory Specification and Generation", *Robotics Research*, 2nd Intl. Symp., Kyoto, 1984, MIT Press, 1985, 187–193.

[14] K. Pingle, R. Paul, R. Bolles (1974)
"Programmable Assembly, Three Short Examples", Film, Stanford AI Lab, October 1974.

[15] M. Peshkin, A. Sanderson (1985)
"The Motion of a Pushed, Sliding Object; Part 1: Sliding Friction", Robotics Inst., Carnegie-Mellon Univ., TR 85-18

[16] M. Peshkin, A. Sanderson (1986)
"The Motion of a Pushed, Sliding Object; Part 2: Contact Friction", Robotics Inst., Carnegie-Mellon Univ., TR 86-7

[17] M. Takano (1984)
"Development of Simulation System of Robot Motion and its Role in Task Planning and Design Systems", *Robotics Research*, 2nd Intl. Symp., Kyoto, 1984, MIT Press, 1985, 223–230.

[18] J. Wittenburg (1977)
"Dynamics of Systems of Rigid Bodies", B. G. Teubner, Stuttgart, W. Germany, 1977.

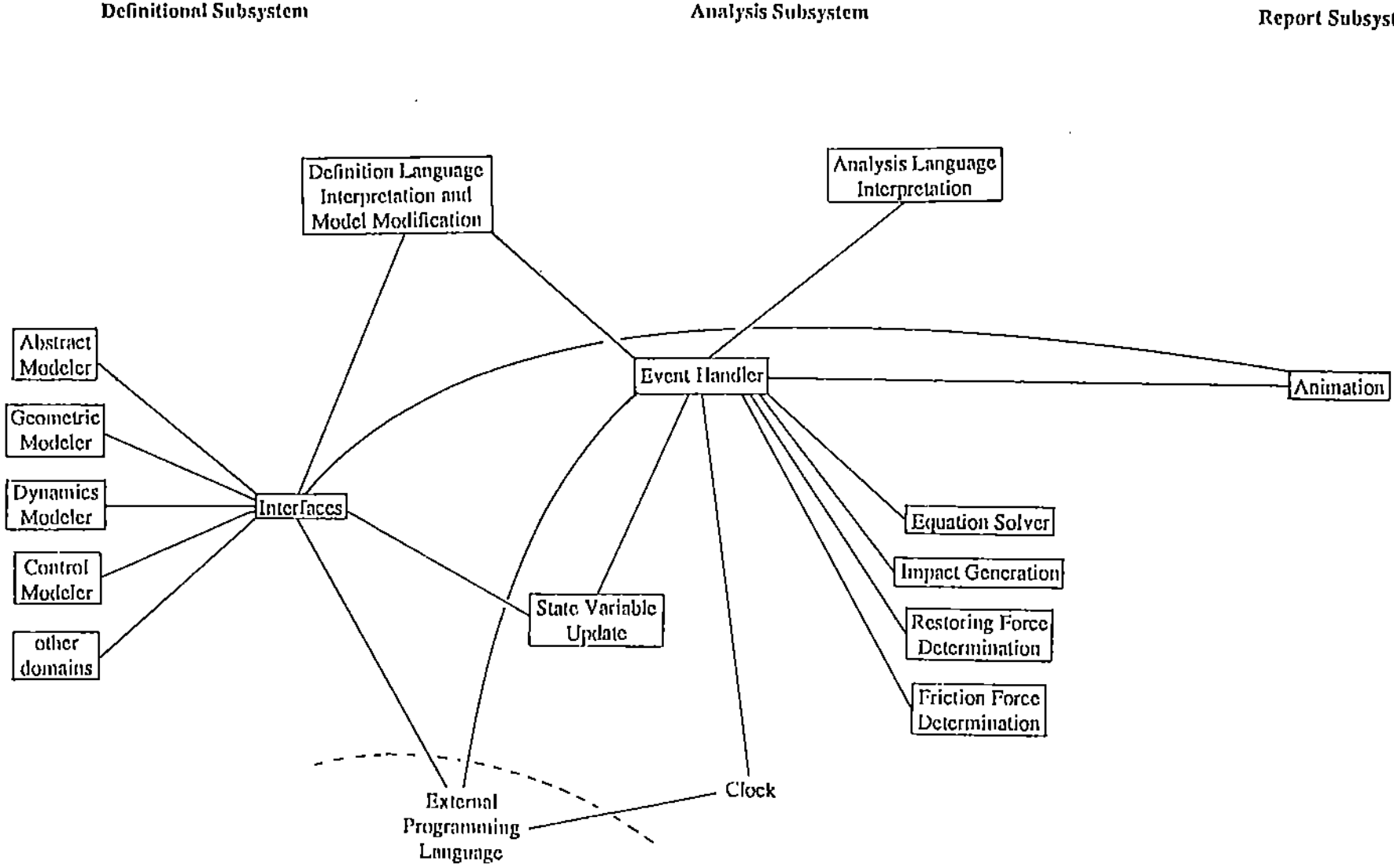
Definitional Subsystem
Analysis Subsystem
Report Subsystem
Definition Language Interpretation and Model Modification
Analysis Language Interpretation
Abstract Modeler
Geometric Modeler
Dynamics Modeler
Control Modeler
other domains
Interfaces
Event Handler
Animation
State Variable Update
Equation Solver
Impact Generation
Restoring Force Determination
Friction Force Determination
External Programming Language
Clock

Figure 2.1

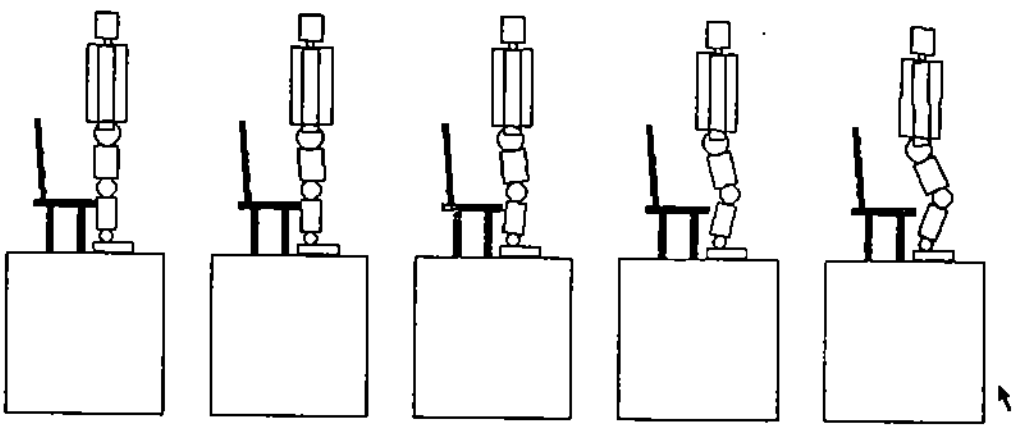

Figure 3.1

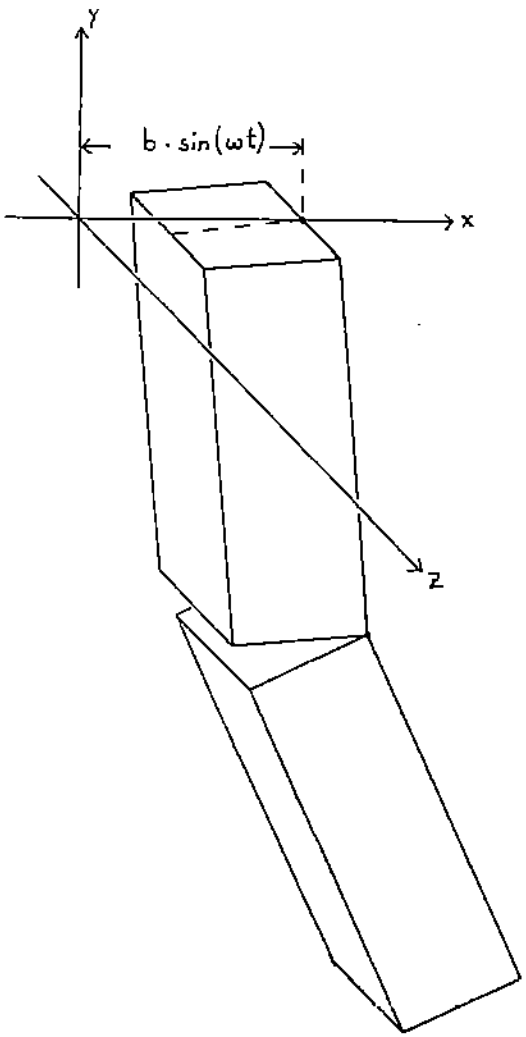

Figure 3.2

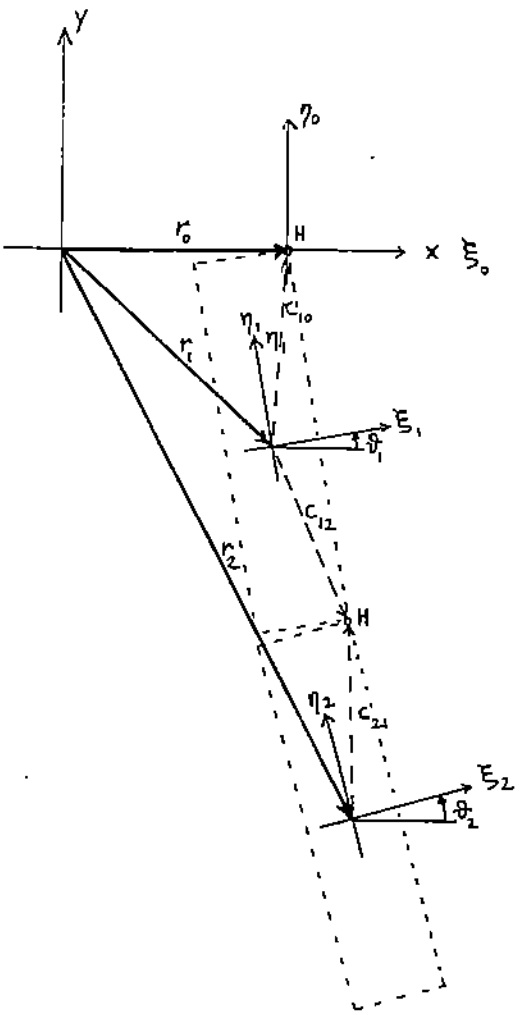
y
η_0
r_0
H
x
ξ_0
r_1
r'_{10}
η_1
η'_1
r'_1
E_1
ϑ_1
c_{12}
r'_{21}
H
η_2
c'_{21}
E_2
ϑ_2

Figure 3.3

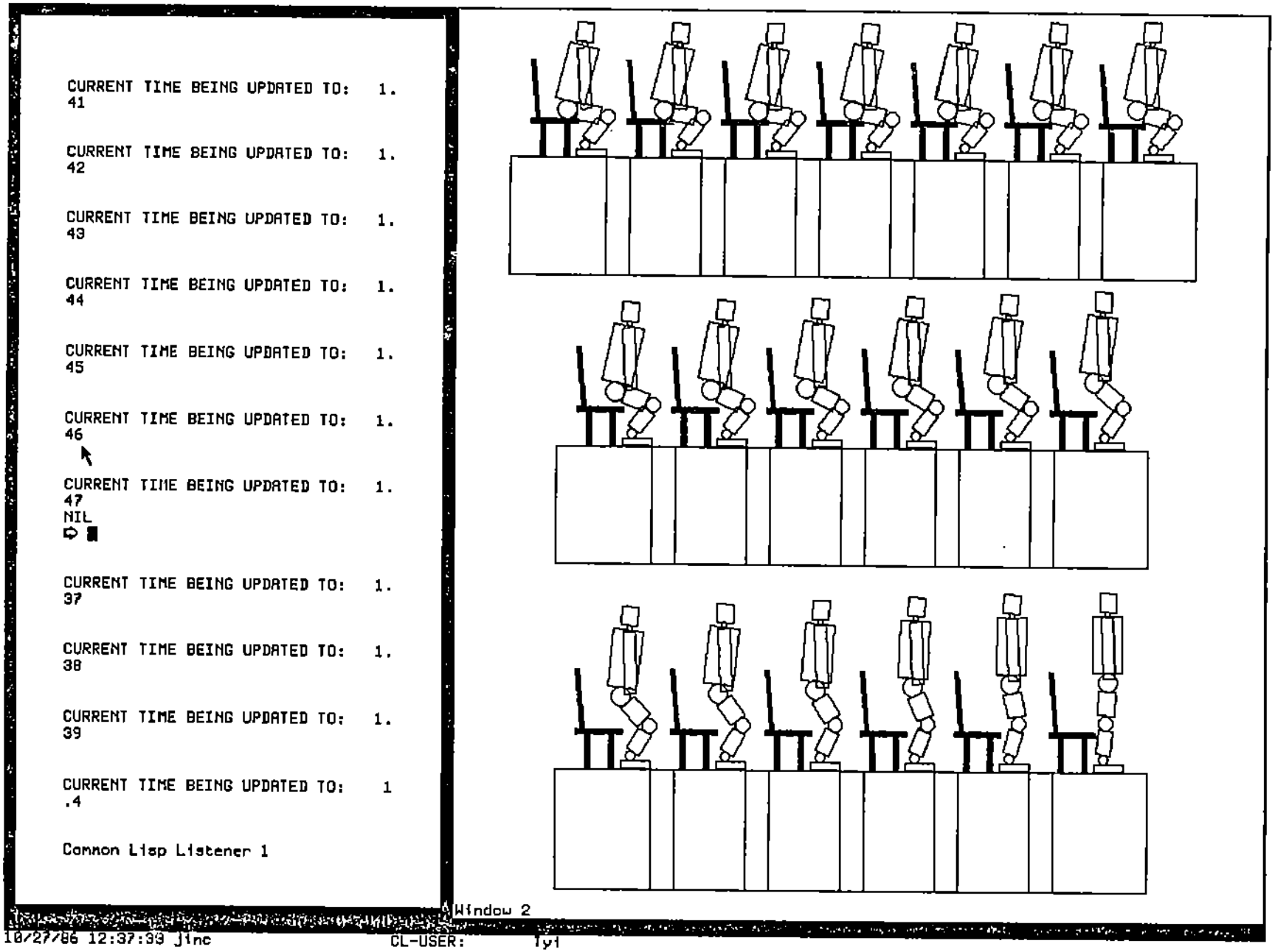
CURRENT TIME BEING UPDATED TO: 1.
41
CURRENT TIME BEING UPDATED TO: 1.
42
CURRENT TIME BEING UPDATED TO: 1.
43
CURRENT TIME BEING UPDATED TO: 1.
44
CURRENT TIME BEING UPDATED TO: 1.
45
CURRENT TIME BEING UPDATED TO: 1.
46
CURRENT TIME BEING UPDATED TO: 1.
47
NIL
CURRENT TIME BEING UPDATED TO: 1.
37
CURRENT TIME BEING UPDATED TO: 1.
38
CURRENT TIME BEING UPDATED TO: 1.
39
CURRENT TIME BEING UPDATED TO: 1
.4
Common Lisp Listener 1
Window 2
10/27/86 12:37:33 jinc
CL-USER: (y1

Figure 4.1

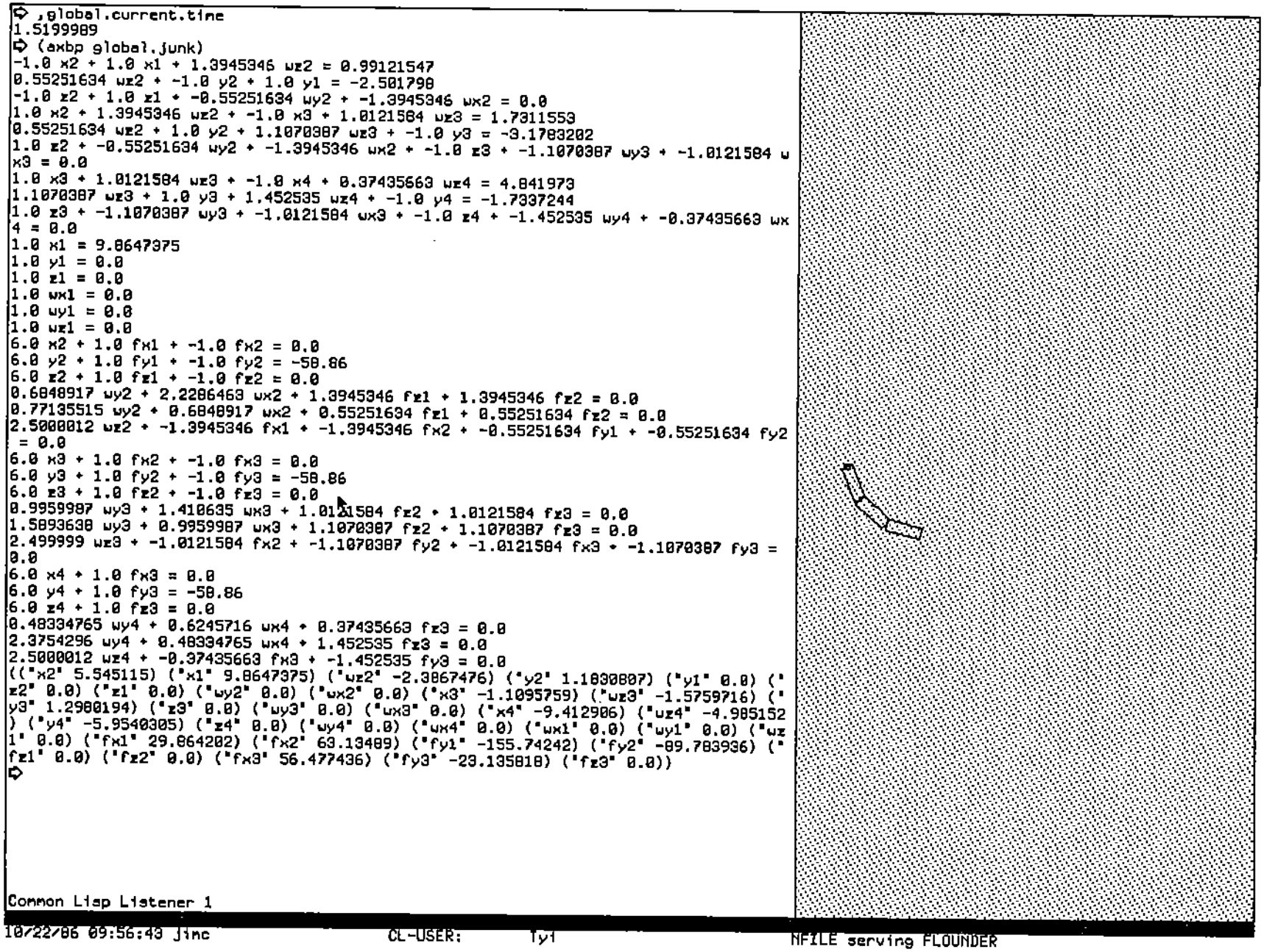
,global.current.time
1.5199989
(axbp global.junk)
-1.0 x2 + 1.0 x1 + 1.3945346 wz2 = 0.99121547
0.55251634 wz2 + -1.0 y2 + 1.0 y1 = -2.501798
-1.0 z2 + 1.0 z1 + -0.55251634 wy2 + -1.3945346 wx2 = 0.0
1.0 x2 + 1.3945346 wz2 + -1.0 x3 + 1.0121584 wz3 = 1.7311553
0.55251634 wz2 + 1.0 y2 + 1.1070387 wz3 + -1.0 y3 = -3.1783202
1.0 z2 + -0.55251634 wy2 + -1.3945346 wx2 + -1.0 z3 + -1.1070387 wy3 + -1.0121584 w
x3 = 0.0
1.0 x3 + 1.0121584 wz3 + -1.0 x4 + 0.37435663 wz4 = 4.841973
1.1070387 wz3 + 1.0 y3 + 1.452535 wz4 + -1.0 y4 = -1.7337244
1.0 z3 + -1.1070387 wy3 + -1.0121584 wx3 + -1.0 z4 + -1.452535 wy4 + -0.37435663 wx
4 = 0.0
1.0 x1 = 9.8647375
1.0 y1 = 0.0
1.0 z1 = 0.0
1.0 wx1 = 0.0
1.0 wy1 = 0.0
1.0 wz1 = 0.0
6.0 x2 + 1.0 fx1 + -1.0 fx2 = 0.0
6.0 y2 + 1.0 fy1 + -1.0 fy2 = -58.86
6.0 z2 + 1.0 fz1 + -1.0 fz2 = 0.0
0.6848917 wy2 + 2.2286463 wx2 + 1.3945346 fz1 + 1.3945346 fz2 = 0.0
0.77135515 wy2 + 0.6848917 wx2 + 0.55251634 fz1 + 0.55251634 fz2 = 0.0
2.5000012 wz2 + -1.3945346 fx1 + -1.3945346 fx2 + -0.55251634 fy1 + -0.55251634 fy2
= 0.0
6.0 x3 + 1.0 fx2 + -1.0 fx3 = 0.0
6.0 y3 + 1.0 fy2 + -1.0 fy3 = -58.86
6.0 z3 + 1.0 fz2 + -1.0 fz3 = 0.0
0.9959987 wy3 + 1.410635 wx3 + 1.0121584 fz2 + 1.0121584 fz3 = 0.0
1.5893638 wy3 + 0.9959987 wx3 + 1.1070387 fz2 + 1.1070387 fz3 = 0.0
2.499999 wz3 + -1.0121584 fx2 + -1.1070387 fy2 + -1.0121584 fx3 + -1.1070387 fy3 =
0.0
6.0 x4 + 1.0 fx3 = 0.0
6.0 y4 + 1.0 fy3 = -58.86
6.0 z4 + 1.0 fz3 = 0.0
0.48334765 wy4 + 0.6245716 wx4 + 0.37435663 fz3 = 0.0
2.3754296 wy4 + 0.48334765 wx4 + 1.452535 fz3 = 0.0
2.5000012 wz4 + -0.37435663 fx3 + -1.452535 fy3 = 0.0
(("x2" 5.545115) ("x1" 9.8647375) ("wz2" -2.3867476) ("y2" 1.1030807) ("y1" 0.0) ("
z2" 0.0) ("z1" 0.0) ("wy2" 0.0) ("wx2" 0.0) ("x3" -1.1095759) ("wz3" -1.5759716) ("
y3" 1.2980194) ("z3" 0.0) ("wy3" 0.0) ("wx3" 0.0) ("x4" -9.412906) ("wz4" -4.985152
) ("y4" -5.9540305) ("z4" 0.0) ("wy4" 0.0) ("wx4" 0.0) ("wx1" 0.0) ("wy1" 0.0) ("wz
1" 0.0) ("fx1" 29.864202) ("fx2" 63.13489) ("fy1" -155.74242) ("fy2" -89.783936) ("
fz1" 0.0) ("fz2" 0.0) ("fx3" 56.477436) ("fy3" -23.135818) ("fz3" 0.0))
Common Lisp Listener 1
10/22/86 09:56:43 jinc
CL-USER:
Tyi
NFILE serving FLOUNDER

Figure 4.2

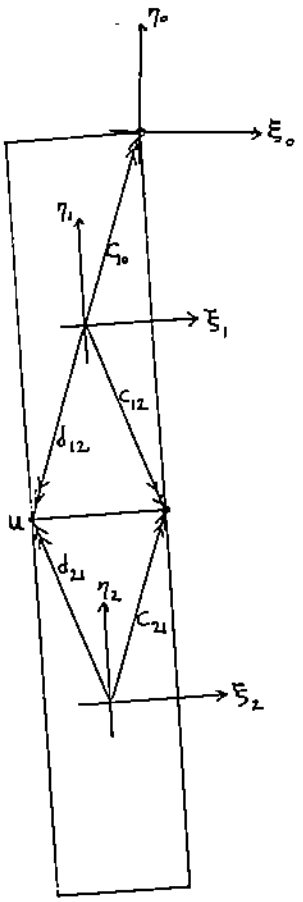

Figure 4.3

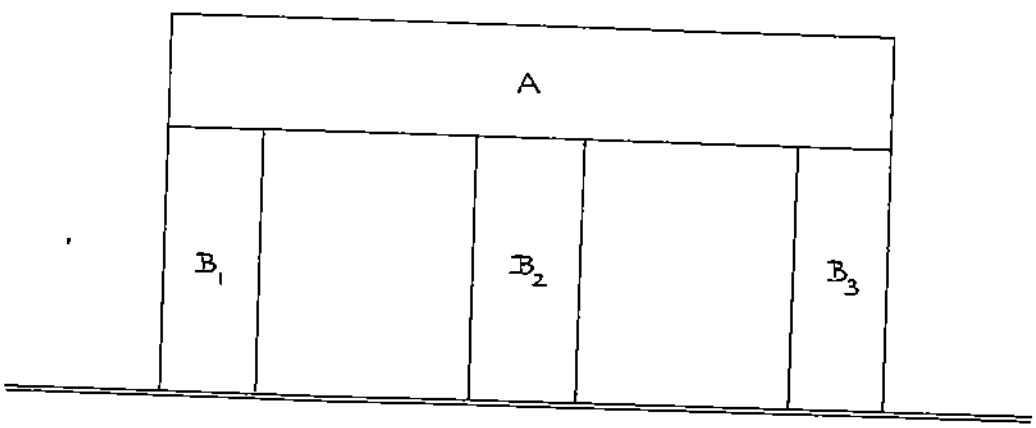

Figure 4.4