1993

Project Isaac: Building Simulations for Virtual Environments

George Vaněček

James Cremer

Report Number:
93-068

PROJECT ISAAC: BUILDING SIMULATIONS FOR VIRTUAL ENVIRONMENTS

George Vanecek, Jr.
James Cremer

CSD-TR-93-068
November 1993

Project Isaac: Building Simulations for Virtual Environments

George Vaněček Jr.* James Cremer†
Purdue University University of Iowa

November 15, 1993

Abstract

In this paper, we introduce the architecture of project *Isaac*. The project was created to develop efficient, robust and flexible physical-based simulations for virtual environments. It is based on the combined experiences of project *Newton*, *ProtoSolid*, and the Iowa driving simulator, developed concurrently at Cornell University, Unversity of Maryland, and Purdue University in the late 1980's. The project will develop a distributed simulation server consisting of a simulation core, a dynamics module, a geometry module based on *Proxima*, a control module and a task manager.

With *Isaac*, it will be possible to simulate virtual worlds populated with autonomous robots under various levels of control. The control could range from simple script-based scenario control to complex artificial intelligence control such as what is now under study by project OZ at Carnegie Mellon University.

The main objective is to provide the scientic basis for and to demonstrate the advantages of a model-driven simulation system that integrates dynamics, geometry and control. This will have a substantial impact on the simulation and virtual reality communities.

*Purdue University, Department of Computer Sciences, West Lafayette, IN 47907-1398. Phone: (317)49-46964, Fax: (317)49-40739, Email: vanecek@cs.purdue.edu

†University of Iowa, Department of Computer Science, Iowa City, IA 52242. Phone: (319)335-0736, Fax: (319)335-0627, Email: cremer@cs.uiowa.edu

Contents

1 Introduction

Research in creating interactive virtual environments has focused largely on walk-throughs and on immersion hardware such as head-mounted displays, graphics software, and devices for human-computer interaction such as gloves. However, many existing virtual environments suffer because their worlds are populated by objects that either users cannot really interact with (e.g. users can only look at them) or that do not behave in physically satisfying ways (e.g. objects released from a hand don't fall, and active non-user entities such as robots are purely animated or scripted). Such shortcommings can now be addressed by physically-based simulations. These simulations can greatly enrich virtual environments and will certainly become an integral part. Yet research on incorporating physical simulations into interactive virtual environments lags behind the other developments. An approach combining the sound technical basis of mechanical engineering work in dynamics simulation with the interactivity and controllability of graphics and animation systems is needed.

Recently, the graphics, animation, and virtual environment communities have shown a lot of interest in physically-based simulation since it provides a means to enhance the believability of their products. In the engineering community, an enormous amount of physical systems simulation research and simulation software development has been carried out. However, existing simulation tools were not designed specifically to support the requirements of virtual environments and, in fact, do not well support them. Dynamics simulation systems from the mechanical engineering domain (e.g. DADS[20],ADAMS[33], NEWEUL[27]) support analysis of mechanisms and machines in a standard paradigm: formulate motion equations and kinematic constraints, and then numerically integrate them over some time period. They do not support control of complex high-degree-of-freedom objects, and do not integrate geometry and dynamics well enough to support n-body collision detection and two-body contact analysis on other than a very rudimentary level. Work in the graphics and animation community has produced software that is somewhat more usable in virtual environments—for instance, they support interactivity and some collision detection techniques. However, the level of sophistication of these systems is not very high. Many are not robust—they are not based on sound, accurate, efficient numerical techniques, and they will not scale to virtual environments of interesting size (e.g. multiple many-legged walking robots interacting in complex geometric environments).

Our goal is to develop a physical-based simulation support consisting of a distributed simulation server, called *Isaac*, that provides an efficient, robust, and flexible simulation base for virtual environments. Our design of *Isaac* partitions five key modules:

- a simulation core that contains state-of-the-art numerical methods and that efficiently and robustly handles *on-line constraint changes.* In virtual environments collisions occur, contact relationships change, and motor control pro-

grams or high-level plans change state. In *Isaac*, these correspond to constraint changes in the underlying equations.

- a dynamics module that is responsible for formulating the motion equations that capture the basic behavior of physical objects and for interacting with geometry to handle collision and contact dynamics.

- a geometry module that efficiently and robustly supports n-body collision detection and two-body contact analysis; also, a geometric database that will manage the global geometric information of a virtual environment to enable such operations as proximity queries and planning.

- a control module that supports high-level specification of motion control (including specification of low-level controllers such as PID controllers for, say, robot joints, as well as higher-level controllers coordinating a high-degree of freedom mechanism such as an anthropomorphic robot) as well as scenario and behavioral control (including coordinating of multiple agents, planning and control high-level agents behavior).

- a task management module that manages the distribution of computations across a set of *Isaac* server processes. The task manager oversees resource allocation, synchronizes computations as necessary, and manages interprocess communication.

These *Isaac* modules support three component crucial to virtual environments—dynamics, geometry, and control. In the following sections, we describe the scientific research problems in the context of the these three components of the *Isaac* system.

2 Background and Historical Perspecive

Before we begin the discussion of the technical basis for the development of *Isaac*, we first outline our background in this area. We take a historical approach and outline the development of *Newton* on whose experience the *Isaac* system is built. This is followed by sections presenting the technical ideas behind *Isaac* and research problems that need to be addressed.

2.1 History of Newton

The roots of the *Isaac* project include a number of projects at Iowa, Purdue, and Cornell. The *Newton* project, conceived by Christoph Hoffmann and John Hopcroft in 1986, was one of the first attempts to integrate geometry and control with dynamics. Its goal was to bring computer-aided *design* and computer-aided *analysis* closer together by providing simulation capabilities based on, and accounting for,

part geometry. One of the driving problems, for instance, was the design of multi-fingered robot manipulators like the Salisbury hand. A system that is to be used to evaluate hand designs must support not only dynamics, but also geometry, to analyze contact during manipulations, and control, to test controllability of the hand design.

Newton[22, 13, 14], written by Cremer and Bouma, was quite successful on some fronts but less so on others. For many basic dynamics problems, it was easy to use. But, it lacked state-of-the-art numerical integration techniques, and was less accurate and robust than commercial dynamics simulators. It was successful as an experimental testbed for collision and contact research. However, the contact and collisions module never reached an acceptably robust and efficient level. One of the primary reasons for this is that the interplay between dynamics and geometry had not been carefully studied before the development of *Newton* and was not well understood. Although it turned out that the *Newton* architecture was not ideally suited to efficient implementation of the results, *Newton* provided an important testbed for research that has led to a clear understanding of dynamics-geometry integration.

Up to about 1989, *Newton* dealt with geometry only as parameterized primitives. The advantages were that *Newton* could ignore the exact boundary of primitives and deal with them exclusively on the parametric level; this allowed Cremer and Bouma to easily program a collision detection and contact analysis module. The disadvantage of using parameterized primitives was that it allowed only a limited class of solids that could be simulated. In 1989, Vaněček brought his *ProtoSolid* solid modeling system [44] to Purdue from University of Maryland where it was developed. Since both *Newton* and *ProtoSolid* were developed in Common Lisp, Vaněček began to study the problem of integrating *ProtoSolid* to *Newton* to provide a broader class of nonconvex polyhedral objects. This was accomplished by packaging *Newton* and ProtoSolid into servers and creating a protocol for communicating geometric event information between the two servers. To control the system, Vaněček wrote a Solid Modeling Interface (SMITool) on a Silicon Graphics as a visual graphics interface. The architecture is shown in Figure 1.

The initial difficulty with this system was that ProtoSolid was not specifically designed to support a dynamics simulation system, but mechanical design. In the first year, ProtoSolid had to be upgraded to perform mass-property computation and to handle collision detection. The later was done by adding Bruce Naylor's Binary Space Partition (BSP) Trees [17, 32]. With the BSP tree support *Newton* could perform simulations with any polyhedra, but only for simple contacts [10]. Complex contact models such as the simulation of the tumbling rings, shown in Figure 2, failed. This was later found to be due to the insufficient contact information obtainable from the BSP support. Specifically, with the BSP trees, only edge information was possible, and as our later paper on contact analysis shows, this was insufficient. Consequently, trying to overcome the inefficiency of the BSP trees, Vaněček generalized the trees to multi-dimensional structures and later to the

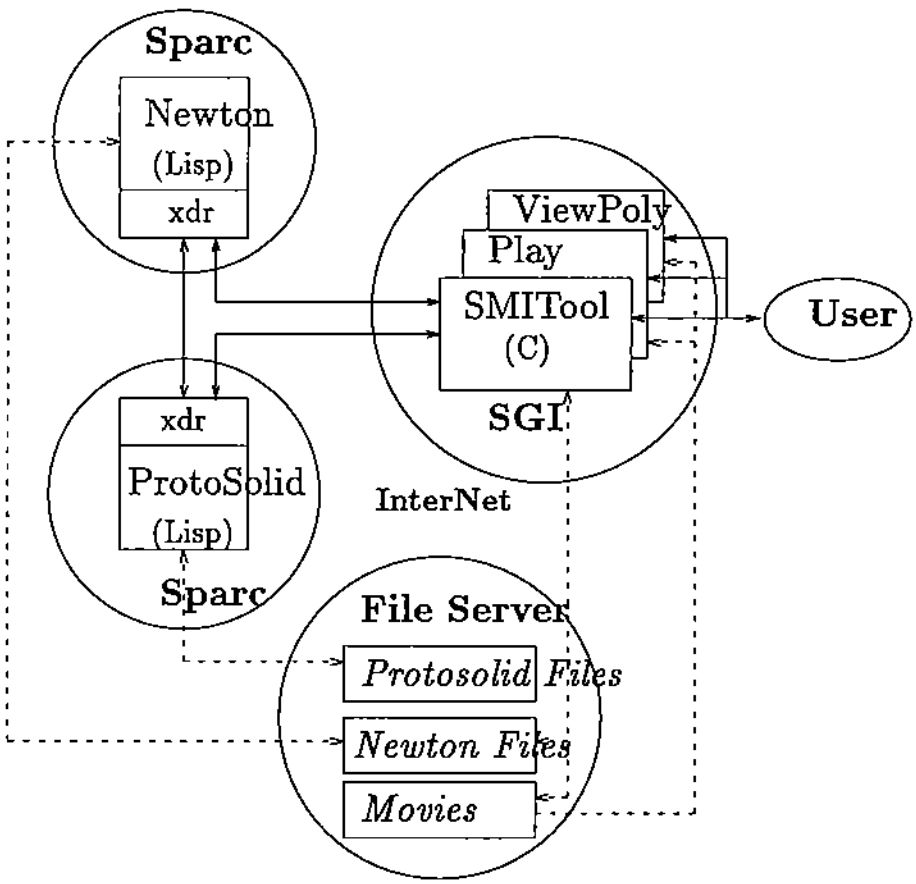

Figure 1: The early architecture of the *Newton/ProtoSolid* simulation system.

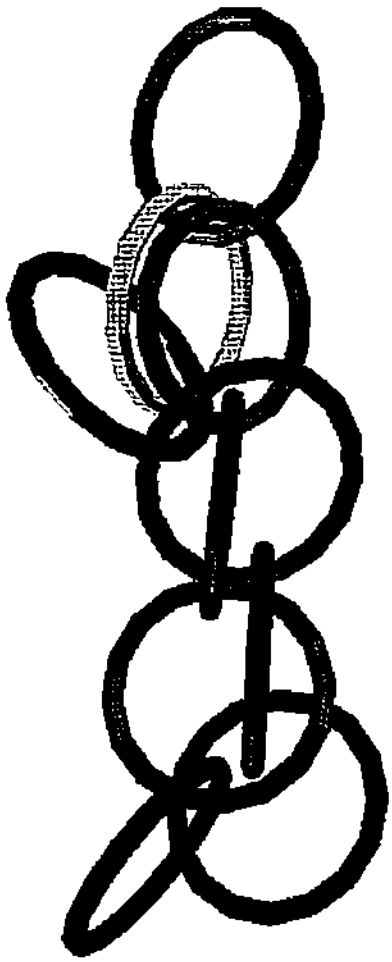

Figure 2: Newton *simulation of tumbling rings. This idea was obtained from an article in "Mathematical Games", Scientific American, 1965.*

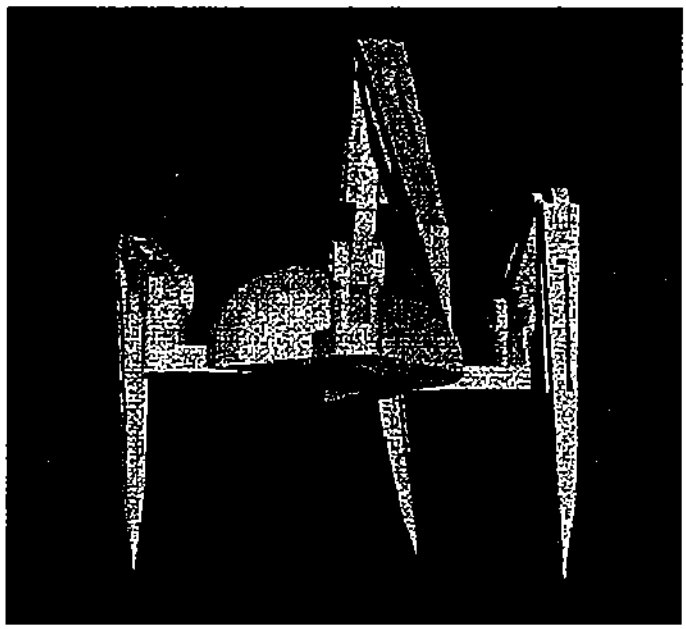

Figure 3: *Controlling GUMBY at Purdue.*

Brep-Index. This led to the model of contact now present in *Newton.* To make this technology widely available, Vaněček started the development of a system called Proxima in C++.

The control portion of *Newton* was the most successful component. Paradigms for *programming* the control of high-degree-of-freedom mechanisms were developed. These applied to both the domains of graphics and animation as well as mechanical engineering and robotics. This work has been influential, for example, in the development of the scenario control subsystem of the Iowa Driving Simulator. At the University of Iowa, work on a language for programming high level control that integrates nicely with the Isaac philosophy has been conducted. Its use has already been demonstrated with *Newton* by students who programmed a one-leg hopper that can hop from place to place and who programmed two linkages holding rackets that play ping-pong on a small table. At Purdue, Bouma has written control programs on top of *Newton* to control the walking of a four-legged robot (see Figure 3), and at Cornell, others have programmed various bipeds and hopping machines to perform complex functions such as standing-up, sitting, walking, jumping and riding a bicycle[25, 34, 41].

In retrospect, *Newton* was a prototype system that clarified a number of not-well-understood difficulties in integrating dynamics and geometry. From these experiences, we have evolved the simulation architecture for Isaac. It supports the robust and efficient integration of geometry and dynamics, detailed in Sections 3.1 and 3.5.

3 Isaac Architecture

A crucial feature of a simulation system for virtual environments is the ability to handle changes: collisions occur; contacts form, remain for a while, and break; motion control algorithms change state; active agents change their goals based on sensed information; and so on. In *Isaac* such changes are signaled by *events.* Handling of events generally consists of changing the set of equations representing object behavior. For example, two initially not-in-contact mechanisms may be modeled by two independent sets of equations (motion equations, kinematic constraints, and perhaps some control equations). If the mechanisms come into contact (and don't immediately break contact - i.e. that don't just bounce away from each other) an equation representing a new kinematic constraint will be added. This equation couples the two previously independent sets of equations. At some later time the contact might break; the equation set would then be modified again.

The design of *Isaac* has two major goals; namely,

- support efficient constraint changes, and
- support modularity and "constraint programming" style of module interaction.

3.1 Simulation Core

From our work with *Newton*, we found that it was especially convenient to view the module interaction in "constraint programming" terms. The *Isaac* architecture makes this explicit. At the lowest level of the system lies the *simulation core.* It is ultimately responsible for solving a set of equations and advancing the simulation through time. The set of equations that the core solves can be viewed as a set of constraints that the other modules — dynamics, geometry, and control — manipulate. When events occur these modules may add, remove, or modify constraints. These higher-level modules are provided with a "constraint programming" view of the simulation. They interact with the simulation core through a simple well-defined constraint manipulation interface. Note that while the interface may be simple to define (e.g. containing a small number of constraint set manipulation routines) it is not a trivial matter to implement it well. The simulation core will contain a variety of equation solving methods. Depending on particular features of a simulation, some methods may be more appropriate than others. For example, for some problems standard DAE solvers like MEXX[31] will be appropriate. For others, especially those involving a significant number of collisions and contact changes, a more specialized integrator such as that outlined in Section 3.5 will be necessary. For efficiency purposes, the constraint programming interface routines will each have a number of implementations based on the various solvers. For example, when MEXX is being used, the "add-equation" method that is implemented in terms of MEXX data structures will be used. If instead, we had a generic set

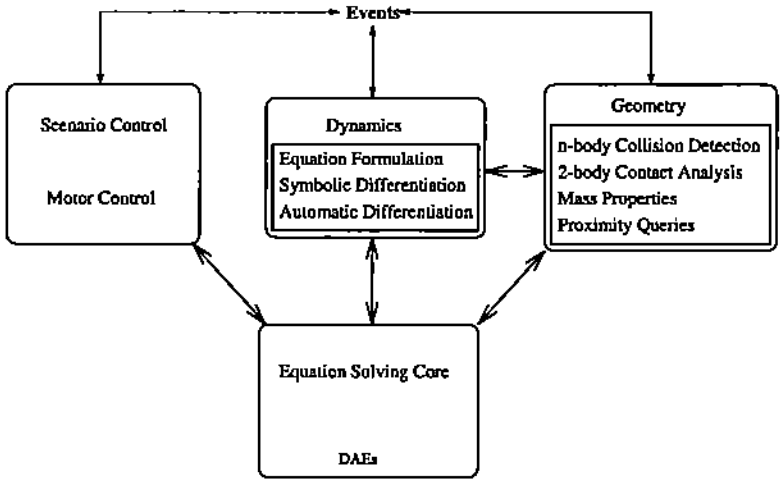

Figure 4: *The* Isaac *architecture.*

of interface routines that worked in terms of some common symbolic equation format, the simulation core would have to translate between that representation and a particular solver's representation at run-time. This would, in general, lead to unacceptable performance.

Within the simulation core lies the event manager. Various *Isaac* modules can define *events.* To define an event, a module specifies how the event is to be *detected* and how it is to be *resolved.* Event detection may correspond to a function value passing through zero or to detection of geometric interpenetration. Event resolution may involve formulating a set of equations representing handling of impact, adding or removing equations corresponding to contact constraint changes, or simply changing gain values within a controller. A number of issues complicate efficient event handling. The time of event occurrence must be *isolated* efficiently, and continuation of the simulation after the event must be done with minimal effect on efficiency and accuracy.

The basic *Isaac* architecture is shown in Figure 4. Each of the dynamics, geometry, and control modules interacts with the simulation core in terms of constraints. Dynamics formulates basic motion equations and kinematic constraints and hands them to the simulation core. During simulation, the dynamics, control, and geometry modules modify the initial equation set by adding and removing equations as warranted by the occurrence of events.

3.2 Dynamics

The dynamics module of *Isaac* is responsible for formulating a set of motion equations and for providing them to the simulation core. It is also responsible for formulating equations (and related mathematical information such as Jacobians) corre-

sponding to kinematic constraints. For mechanisms involving only *permanent* kinematic constraints, those corresponding to standard physical joints such as revolute joints, a variety of standard dynamics formulations will be used. The basic formulation will use a maximal set of Cartesian coordinates, in the style of Haug/DADS[20] and Cremer/Newton[13]. Such formulations are particularly amenable to specifying and implementing constraint changes. As development of *Isaac* proceeds and as efficiency considerations require, other formulations, such as the recursive formulations developed by Haug and colleagues at Iowa[1, 2, 3, 43], will be introduced.

When contact constraints, called *temporary constraints*, are present, the dynamics module interacts with the geometry module in order to formulate the appropriate set of equations[1]. As described in Section 3.5, a contact constraint is modeled using two sets of inequalities: one for the dynamics portion of the constraint and one for the geometric portion of the constraint.

The dynamics module is also responsible for defining and handling events that represent special dynamics circumstances. In particular, for a temporary constraint, there is a force condition defining when the corresponding contact will break. The dynamics module is responsible for specifying how to detect the occurrence of contact breakage and what to do about it (e.g. how to update the equations to a consistent state once more).

3.3 Geometry

As already introduced, a virtual-environment simulation requires a geometric support that provides four major components. These are

1. the representation of the geometry of the environment which takes into account moving objects, fixed objects such as the floor, and proximity queries,
2. the determination of mass properties of the movable solid objects,
3. fast n-body collision detection, and
4. fast two-body contact analysis.

In this section we describe these geometric components in detail.

3.3.1 Representing the Geometric Environment

Geometrically, the environment can be thought of as being composed of objects that are assumed inpenetrable and nondeformable. The assumption of inpenetrability is necessary to create a realistic environment—physical objects do not interpenetrate. The second assumption is not. It is used now to limit the complexity of the problem. These two assumptions dictate the types of representations used.

[1]Throughout the proposal, we use the term *equations* to include inequalities as well as true equations.

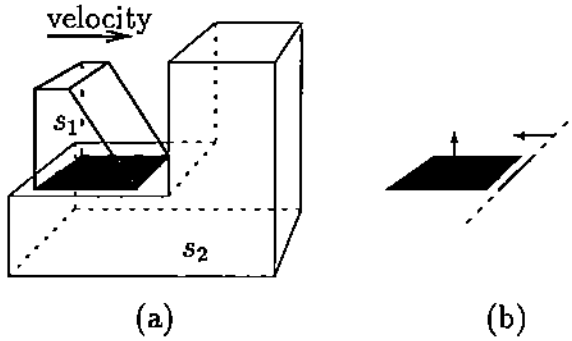

Figure 5: *An example where a two objects in temporary-contact will also collide. (a) show the objects at time t, and (b) shows the two regions at time $t + \Delta t$, one for the temporary-contact, the other for the collision.*

We partition the objects into two categories: movable objects and fixed, immovable objects. Objects that are immovable typically represent walls, the ground, or a frame around a window. Objects that move are things like a chair, a lever in some mechanism or a robot under control. The differences between these categories are their extents and their dynamics. While objects that move must be represented as closed volume objects, the fixed objects can be relatively large, oriented lamina (i.e., surfaces) and not closed volumes. Because, in terms of the dynamics, the objects that do not move need not have their mass properties computed or motion equations formulated, all fixed objects can be combined geometrically into a single complex object. This way, there is only one fixed object. All other objects are movable. The only requirement for the fixed object is that in terms of its oriented surfaces, all moving objects remain on one side of the surface, namely the outside.

Initially, we will model all objects as planar polyhedra. Later, we propose to add free-form surfaces, as described in Section ??.

3.3.2 Mass Properties

For objects to move, the dynamics module must formulate the motion equations using the mass properties of the objects. This consists of computing the inertial matrix which encodes the moments of inertia of the object. Since the objects are assumed rigid, and thus no deformations take place during collisions, this inertial matrix does not change during a simulation and can be precomputed. This is a straight forward problem for which efficient boundary-based algorithms exist.

3.3.3 Collision Detection

Our model-driven dynamics simulation paradigm depends exclusively on the automatic detection of collisions between objects. By collision detection, we mean *the detection of objects in close proximity.* Note that two objects that are already in contact, such as a book on a table, may also collide, such as when the book falls over.

To illustrate this, consider Object s_1 of Figure 5 sliding on top of Object s_2; at some later time, s_1 collides with the vertical inside wall of s_2. This subsequent collision is handled as a contact detection and analysis problem. From the geometry alone it is not possible to distinguish a contact from a collision. Thus once two objects come into contact, the collisions and contact have to be analyzed simultaneously. Knowing when to do this more detailed analysis is the problem of collision detection.

Since all collision events stop time to change the equation motion velocities, and this can happen quite frequently, the detection algorithm must be very fast. For objects that are far apart, the exact geometry of the boundary is not important. For this reason and for computational benefits, we approximate objects that are far apart by their convex hull. We then check their proximity using the fastest known algorithm for convex object, the Lin and Canny's algorithm [28]. However, we have to handle n objects simultaneously. Even this fast two-body collision detection algorithm requires $O(n^2)$ time if it performs pair-wise collision checks, and this is prohibitive if large number of moving objects are simulated.

There are a number of algorithms that address the n-body problem. For instance, Lin, Manocha and Canny give a simple extension of the Lin and Canny's algorithm for convex objects by estimating the possible time of collision [29]. Similar extension was proposed by Dwarkin and Zeltzer at MIT which uses simple time-parameterized trajectories of the objects to predict possible intersections [15]. In both cases, the predicted times are placed into a time-prioritized queue. The simulation then continues until the time of the first event on the queue, at which time the two objects indicated by the event are checked. The simulation then continues until the next event on the queue. These approaches assume that the number of collisions in any given time interval is small, the trajectories of all objects are known a priori and that objects are relatively far away from each other. These assumptions cannot be made in our model-driven simulations. Firstly, there may be a large number of closely spaced collisions, and secondly, we cannot predict the trajectory of objects which are under motion control. Robots' motion is unpredictable, thus the potential collision of a robot with another object cannot be generally predicted computationally.

In our case, an incremental algorithm based on the original Lin and Canny algorithm, using coherence between frames and Voronoi partitions is better. This approach would allow a constant number of updates per object bounded by the number of adjacent objects. This is linear, rather than the naive algorithm's n^2 updates. Both Vaněček and Manocha, as well as others, are working on an the n-body problem. Our approach will use the modified Lin and Canny to check for intersection of the convex-hulls of the objects. When the convex hulls intersect, a more detailed algorithm will be used to detect collisions and contact. This is now described.

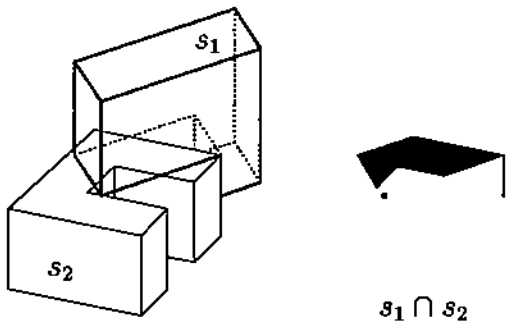

Figure 6: *Two objects in contact, and their three contact regions resulting from a set-theoretic intersection.*

3.3.4 Contact Analysis

Contact analysis is a process that provides a detailed description of the two-body contact regions. Note that the property that objects cannot interpenetrate is not supported by the representations of the objects. Regardless of the representation—BRep, CSG or Octrees, for instance—there is no inherent support for disallowing their interpenetration. The property must be supported computationally. The dynamics module does this by adding constraint equations to the set of motion equations which reduces the degrees of motion freedom. The contact regions are converted to equations that describe where forces are applied to the geometric limits of the region to keep the two objects from interpenetrating.

Bouma and Vaněček have shown that the contact analysis requires a full set-theoretic intersection of the two objects to determine the contact regions, followed by the analysis of the contact regions [11]. For example, Figure 6 shows two objects in contact and the set of contact regions obtained from the set-theoretic intersection.

This can be done easily in $O(N^2)$ time where N is the number of vertices, edges and faces in both objects. However, this can be very slow for large N. To speed up this analysis, Vaněček has formulated two helpful techniques. The first is the Brep-index and the second is the back-face culling technique. These are described below. Basically, the analysis begins by culling the vertices, edges and faces that are know a priori not to be in contact and then classifying the unculled vertices, edges and faces of one object against the Brep-index of the second object. For efficiency, we check the Brep of the smaller object against the Brep-index of the larger object. The topological entities that lie on the boundary of the other object are retained and combined into contact regions. For robustness, only the Brep of one object is checked against the other. This alleviates classic robustness problems found in geometric modeling systems [21].

The Brep-Index

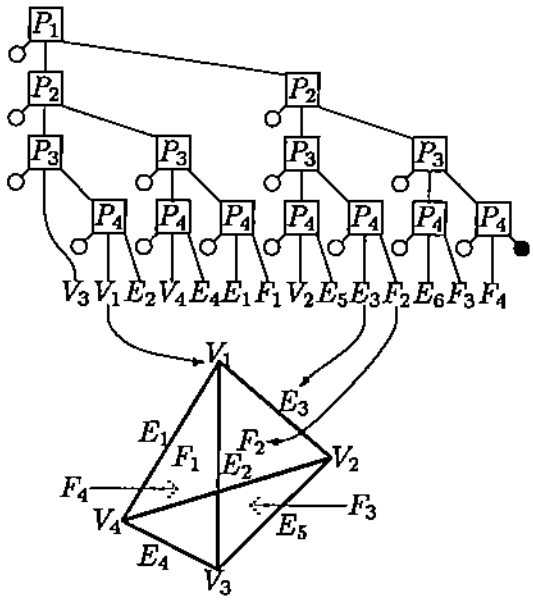

Figure 7: *Example Brep-index for a tetrahedral solid. The white circles indicate outside regions; the black circle indicates the single inside region.*

Contact analysis is based on analyzing contact regions, thus it is boundary based. However, the classification that obtains the contact regions is inherently spatial, not boundary based. For this reason, Vaněček developed a spatial representation of an object that recursively subdivides space into open halfspaces called a multi-dimensional space partitioning (MSP) tree. It is a direct extension of Naylor and Fuchs' BSP tree. The MSP structure allows for a fast search that quickly convergence to the region containing the query point, line segment or polygon. To gain the benefit of both the efficient spacial search and the detail of the boundary, Vaněček combined the MSP tree and the BRep to yield a single unified representation for objects. This representation is called the Brep-index. An example is shown in Figure 7.

Back-Face Culling

In the above section we explained that to find the contact regions, we have to classify all the faces, edges and vertices of one of the objects against the Brep-index of the other object. Although this already reduces the cost of classification from quadratic to subquadratic, we can reduce the cost even more by eliminating roughly half the boundary from needing to be checked. That is, we can cull roughly half the vertices, edges and faces of the Brep if we take the object's relative-velocity into account.

Vaněček has applied the well know back-face culling problem in computer graphics area to that of the contact analysis by using relative velocity instead of the view direction used in computer graphics. This way, a face, for example, can be ignored (i.e., it is known a priori not to collide) when the relative-velocity vectors of the points on the face are all pointing in the opposite direction of the normal. We say

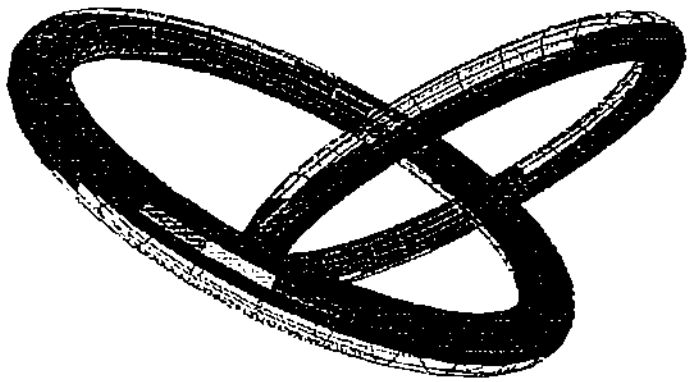

Figure 8: *Two moving, spinning objects with culled faces shown as wire frames.*

that the face is moving backwards and thus cannot collide with anything above it. As an example, refer to Figure 8 showing two moving torii; the faces moving forward are shaded and the ones moving backwards are drawn as wire-frames.

Combining the back-face culling technique with the brep-index yields a very efficient technique that detecting collisions for objects in close proximity and to determine the contact regions for touching objects.

3.3.5 Proxima

Proxima is a set of C++ routines provided as a library, and intended to provide the geometric support described in this section [42]. The primary representation for objects is the BRep. The MSP tree and the Brep-index is the secondary representation. Through these representations, *Proxima* provides a wealth of low-level geometric operations to query the boundary, classify entities, and to obtain mass properties. Presently, *Proxima* does not contain contact analysis nor collision detection.

3.3.6 On the Geometric Complexity

On first inspection, it may appear that the geometric support is unneccessarly complex. This complexity is, never-the-less, inherent in the need to have a well integrated spatial and boundary representation, and the need to support consistent and robust operations in interactive times. Although there are other possible variations on the data structures and algorithms, our particular choices of these are based on what we feel are the best of the current state-of-the-art.

3.4 Control

The ability to control, direct, and choreograph the activities and behaviors of complex active entities is an essential ingredient of a virtual environment system. Here, we make a somewhat hazy distinction between *motion control* and *scenario control.* Motion control consists of specifying and implementing the control of mechanisms in physical terms — i.e. motion control typically consists of specifying joint torques, forces, and accelerations, or constraints on such quantities. Motion control can be quite complex and can involve significant programming in terms of control events that dictate when control parameters or constraints should change. We include in motion control such basic control mechanisms as PD and PID joint controllers. Less clearly in the realm of motion control are programs that control an anthropomorphic robot to walk.

Scenario control consists of higher level controlled activity of simulated entities. It can include AI-style planning activities, the results of which activate appropriate motion control programs. It also includes the coordinating, directing, and choreographing of the activities of multiple simulated entities in accordance with the goals of the scenario author. Virtual environments will have to be flexible and provide a means for a person (either the VE designer/builder, an experimenter, or even the user) to mold the scenario to fit their needs. One person will want four robots behaving and interacting with the user in a particular way, while another will want some different number of robots doing substantially different things. These issues are described in the context of the experiment authoring for the Iowa Driving Simulator in [4, 7].

Isaac is being designed to support both motion control and scenario control. As described in the Section 3.1, the control module will interact with the simulation core in a constraint-programming style. At the lowest level control programs correspond to time-varying sets of constraints, with control events determining the constraint set modification times. At the user-level control programs will be specified in a framework based on Cremer, Kearney, and Hansen's previous work [25, 18] on control for mechanical simulation and on related work by others[19, 12, 39]. The framework is based on a notion of concurrent, hierarchical state machines and is currently being developed by Cremer and Kearney for the Iowa Driving Simulator. We also intend to integrate our work on control of mechanical simulations with the work of Joseph Bates and the OZ project at CMU. See Section 3.7.2 for discussion of this extension to our basic research.

3.5 Integrating Geometry and Dynamics

To achieve efficient and robust simulations, the dynamics and geometry components of the system must be integrated with great care. In particular, the roles of geometry and dynamics components in contact analysis must be well-defined. Consider, for example, a block sliding down an inclined table. Suppose, for simplicity, that the block is oriented so that just one of its corners is in contact with the table. The

point-on-face contact is modeled using (1) an inequality that constrains the point to be on or above the plane of the table, (2) a force condition that says that the contact remains only so long as the reaction force is non-tensile, and (3) conditions indicating that the contact remains only as long as the point remains within the geometric bounds of the table top (the bounds of the face). We distinguish two ways in which the contact can break. One involves a force pushing up on the block such that it breaks contact by lifting off the face. This is a dynamics event — it arises because the non-tensile reaction force condition cannot be consistently maintained. The second type of contact breakage involves the block sliding off the end of the table. This is a geometric event corresponding to violation of the conditions about face bounds.

In early versions of *Newton*, these types of events were not carefully discriminated. In the given example, the geometry module would check at each time instance to see if the objects were in contact or not. If the objects had been in contact at one time instant, but the geometry module determined that at the next time instant they were not in contact, an event would be generated and the contact constraint would be removed. This is, in fact, the wrong thing to do in many cases. Numerical integration can only maintain contact within some prescribed tolerance. It is difficult, at best, to maintain the complete consistency between the dynamics and geometry modules' tolerances that would be required for a geometry-based decision in this situation to be guaranteed to be correct. Precise consistency is, however, not necessary for handling dynamics-type contact events. Unless the contact has reached the face boundary, it can only be broken by the inability to maintain the force condition. Thus, it does not matter if the contact exists or not from the point of view of the geometry module; dynamics can and should make the decision. On the other hand, the second type of contact breakage, that involving the block sliding off the end of the table, does correspond to a geometric event. When it is geometrically determined that the point has reached the face boundaries, a geometric event is signalled and the contact analysis routines analyze the situation and update the equations with new contact constraints. The integration of dynamics and geometry, described informally here, will be described in detail in a forthcoming paper[9]. The basic issues are now reasonably well understood but significant research problems remain, particularly in the area of robustness and efficiency.

To most efficiently support our model of dynamics-geometry integration, a novel numerical integration technique is required. Historically, multibody dynamics simulation programs have relied on ordinary differential equation (ODE) integrators with additional code wrapped around them to allow them to accurately handle differential-algebraic equations systems (DAEs). Recently, integrators designed especially for differential-algebraic equations have become available (see, e.g., [40]). At the outset, *Isaac* will include numerical integration techniques similar to those used in existing state-of-the-art dynamics simulators. Currently, however, Cremer has been working with Florian Potra (an expert on DAEs and a joint Math/CS faculty member at Iowa) and Jeng Yen (an Applied Math Ph.D. from Iowa and the lead

numerical integration person at CADSI, Inc., the developers of DADS) to develop a new DAE integrator that specifically meets the needs of systems like *Isaac*. In particular, it is being designed to efficiently and robustly handle changing constraint sets and constraint sets including inequalities. The method will extend recent work by Potra[36, 37, 38] and Yen[48, 49].

3.6 Distributed Computations

The geometry, dynamics, control, and simulation core components of *Isaac* will all be developed with state-of-the-art efficiency in mind. However, a system such as *Isaac* will naturally benefit from a distributed system organization. Thus, at the top level *Isaac* will consist of a set of *Isaac* simulation-server processes managed by a *task management* process. Each *Isaac* process will be a self-contained simulation server. The entire computation *could* be done within any single process. However, it will be the combined responsibility of the server processes and the task manager to distribute computations across multiple *Isaac* processes. For example, at a fairly simple level, there would be one *Isaac* process responsible for simulating each independent (e.g. kinematically independent) mechanism. Each process would acquire updated state information about other objects as needed. In this model, each process has a local cache containing the complete simulation state, but it is only responsible and authorized to manipulate the components of the state corresponding to its assigned object. Such information replication and implied state communcation is reasonable within *Isaac* because the anticipated numbers of entities and the kinds of information that need to be transferred are relatively small (i.e. not on the scale of systems like SimNet).

There are a number of models for setting up distributed systems. On this, we plan to collaborate with Jim Purtilo at the University of Maryland and his Polylith project. Vaněček has worked with several people involved in the Purtilo's group and continues to maintain a close tie with the University of Maryland's computer science department.

3.7 Applications of Isaac

Isaac is the core virtual environment systems. As it is described in the previous sections, it can be extended and applied to support environments. Looking beyond this the core, there are several extensions we hope to pursue. Here we list some of these.

3.7.1 Generalizing to Free-Form Surfaces

The initial system will use only objects with planar surfaces. One obvious extension is to generalized the objects to curved surfaces.

Based on theory of geometric continuity, various methods have been develop over the past 30 years which construct smooth surfaces. Of these, free-form surface

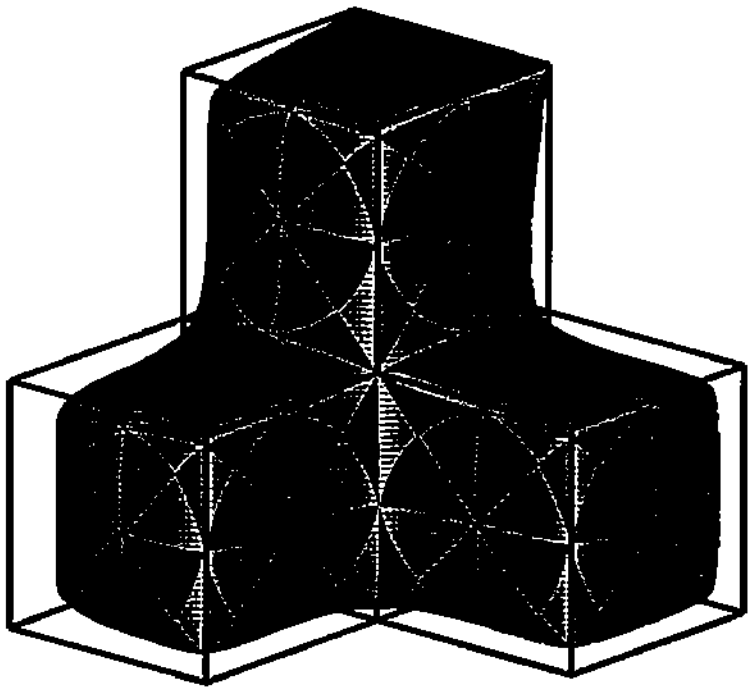

Figure 9: *Polyhedral object smoothed by the free-form surface model of Jörg Peters.*

splines can easily smooth out polyhedral objects. The surfaces would lie in the convex hull of the local underlying geometry. As an example, Figure 9 shows a polyhedron and the smooth surface. The tightness is easily controllable, ranging from the polyhedral mesh (as seen on the top back edge in the figure) to a very smooth object (as seen on the lower edges). Such free-form surface model has been developed by Jörg Peters at Purdue [35]. His work would nicely extend the polyhedral representation of *Proxima.*

3.7.2 Adding Artificial Intelligence

At Carnegie Mellon University, Joseph Bates has been leading the Oz Project. This inter-disciplinary group of AI researchers and artists studies methods by which we might create interactive simulated worlds with the richness and impact of traditional story-telling media, such as film and theater [5]. One important aspect of the Oz work is creating "believable autonomous agents"—real-time, reactive creatures that are sufficiently engaging to maintain a person's interest for as long as needed in the simulated world [6, 30]. These ideas are being actively discussed in the AI community, such as at the AAAI Spring Symposium on Believable Agents, in March 1993, which Bates is organizing (along with Nils Nilsson and Barbara Hayes-Roth of Stanford, and Brenda Laurel of Interval Research). Another central part of the Oz work is how to gently manipulate a running simulation so that it achieves the designer's long term "dramatic" purposes, such as teaching, terrifying, or motivating the user(s), without the experience feeling forced or otherwise unreal [26].

This work complements well the effort at Iowa on the Iowa Driving Simulator [7]. Other drivers, pedestrians, and animals are autonomous agents that need to be engaging and believable to provide a realistic driving experience. The Iowa work on Scenario Control, which concerns the broad progress of the scenario through time, is related to the Oz study of methods for dynamically guiding an experience to achieve maximal dramatic impact. We hope to work with the Oz project to integrate their ideas and technologies into our virtual world framework.

4 For More Information

Many of the papers referenced here as well as related sounds, images, movies and project slides can be found on the World-Wide-Web via XMosaic at

`http://www.cs.purdue.edu/people/vanecek`

This information is kept up to date providing an immediate overview of the project to anyone connected to Internet. We are establishing a document maintained jointly by Purdue and Iowa for *Isaac* leading to all related materials.

References

[1] D.S. Bae and E.J. Haug, "A Recursive Formulation for Constrained Mechanical systems, Part I — Open Loop", *Mechanics of Structures and Machines*, 15(3), 359–382, 1987.

[2] D.S. Bae and E.J. Haug, "A Recursive Formulation for Constrained Mechanical systems, Part I — Closed Loop", *Mechanics of Structures and Machines*, 15(4), 359–382, 1987.

[3] D.S. Bae, R.S. Hwang, and E.J. Haug, "A Recursive Formulation for Real-time Dynamic Simualation", *Proceedings of the 1988 ASME Design Automation Conference*, 499–508, 1988.

[4] M. Bartelme, M. Booth, J. Cremer, D. Evans, J. Kearney, and R. Romano. "Experiment Authoring for Virtual Driving Environments", *Proceedings of the First Eurographics Workshop on Virtual Environments*, Barcelona, Spain, 1–15, Sept 1993.

[5] J. Bates, *Virtual Reality, Art, and Entertainment*, PRESENCE: Teleoperators and Virtual Environments, 1(1) 133–138, 1992.

[6] J. Bates and A. B. Loyall and W. S. Reilly, *Integrating Reactive, Goals and Emotion in a Broad Agent*, Proceedings of the Fourteenth Annual Conference of the Congitive Science Society, Bloomington IN, July 1992.

[7] M. Booth and J. Cremer and J. Kearney. "Scenario Control for Real-Time Driving Simulation", *Proceedings of the Fourth Eurographics Animation and Simulation Workshop*, Barcelona Spain, 103–120, Sept 1993.

[8] W. Bouma and G. Vaněček, Jr. "Collision Detection and Analysis in a Physical Based Simulation," *Proceedings of the Eurographics Workshop on Animation and Simulation*, 191–203, September 1991.

[9] W. Bouma, J. Cremer, and G. Vaněček, Jr. "Robust Efficient Integration of Geometry and Dynamics for Contact Analysis in Multibody Simulation", in preparation, November 1993.

[10] W. Bouma and G. Vaněček, "A Data Structure for Analyzing Collisions of Moving Objects", *IEEE Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, Kailua-Kona Hawaii, Vol I, pp 671–680, January 1991.

[11] W. Bouma and G. Vaněček, Jr. "Contact Analysis in a Physical Based Simulation," *ACM Symposium on Solid Modeling and Applications*, Montreal Canada, May, 1993. (A revised paper to appear in the Journal Computer-Aided Design.)

[12] R. A. Brooks. A robot that walks: Emergent behaviors from a carefully evolved network. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 692–696, May 1989.

[13] J. F. Cremer, "An Architecture for General Purpose Physical System Simulation–Integrating Geometry, Dynamics, and Control." Ph.D. Thesis, TR 89–987, Department of Computer Science, Cornell University, April 1989.

[14] J. F. Cremer and A. J. Stewart. "The Architecture of *Newton*, a General-Purpose Dynamics Simulator", *IEEE International Conference on Robotics and Automation*, pages 1806–1811, 1989.

[15] P. Dworkin and D. Zeltzer, "A New Model for Efficient Dynamic Simulation", Proceedings of the Fourth Eurographics Animation and Simulation Workshop, 135–147, Sept. 1993,

[16] J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes. *Computer Graphics, Principles and Practice*, Second Edition, Addison-Wesley, 1990.

[17] H. Fuchs and Z. M. Kedem and B. F. Naylor, "On Visible Surface Generation by A Priori Tree Structures", *Conference Proceedings of SIGGRAPH 1980*, 14(3) 124–133, Seattle, July 1980.

[18] S. A. Hansen, "Conceptual Control Programming for Physical System Simulation", Ph.D. thesis, Computer Science Department, University of Iowa, May, 1993.

[19] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[20] E. J. Haug. *Computer Aided Kinematics and Dynamics of Mechanical Systems, Volume I: Basic Methods*, Allyn and Bacon, 1989.

[21] C. M. Hoffmann, "Geometric and Solid Modeling", Morgan Kaugmann, San Mateo, Cal, 1989.

[22] C. M. Hoffmann and J. E. Hopcroft. "Simulation of Physical Systems from Geometric Models", *IEEE Journal of Robotics and Automation*, RA–3(3):194–206, June 1987.

[23] C. M. Hoffmann and J. E. Hopcroft. "Model Generation and Modification for Dynamic Systems from Geometric Data," *CAD Based Programming for Sensory Robots*, F50:481–492, 1988.

[24] D. S. Kay and J. T. Kajiya, "Ray Tracing Complex Scenes," *Computer Graphics*, 20(4):269–278, (Proc. SIGGRAPH '86).

[25] J. K. Kearney, S. Hansen, and J. F. Cremer. "Programming mechanical simulations," *The Journal of Visualization and Computer Animation*, April-June, 1993, 113–129.

[26] M. T. Kelso and P. Weyhrauch and J. Bates, "Dramatic Presence", *PRESENCE: Teleoperators and Virtual Environments*, 2(1) 1993 (to appear).

[27] E. Kreuzer and G. Leister. *Programmsystem NEWEUL'90*, Anleitung AN-23, Institut B für Mechanik, Universität Stuttgart, 1991

[28] M. C. Lin and J. F. Canny. "Efficient algorithms for incremental distance computation". Proceedings of the *IEEE Conference on Robotics and Automation*, 1991.

[29] M. C. Lin and D. Manocha and J. Canny. "Fast Collision Detection between Geometric Models", Technical Report TR93-004, Department of Computer Science, University of North Caroline, Chapel Hill, January 1993.

[30] A. B. Loyall and J. Bates, *Real-time Control of Animated Broad Agents*, Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society, Boulder CO, June 1993.

[31] Ch. Lubich, U. Nowak, U. Pöle, and Ch. Engstler, "MEXX — Numerical software for the integration of constrained mechanical systems", Konrad-Zuse-Zentrum für Informationstechnik Berlin, Technical Report SC 92-12, 1992.

[32] B. F. Naylor, *A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes*, Ph.D. Thesis, University of Texas at Dallas, May 1981.

[33] N. Orlandea, *ADAMS (theory and applications)*, Vehicle Systems Dynamics, 16(1987), 121–166.

[34] D. Pai. "Least Constraint: A Framework for the Control of Complex Mechanical Systems", *Proceeding of the Americal Control Conference*, Boston, MA, 1991, 1615–1621.

[35] J. Peters, "Smooth Free-form Surfaces over Irregular Meshes generalizing quadratic splines", *Computer Aided Geometric Design*, 10, 347–361, 1993.

[36] F. A. Potra, "Runge-Kutta Integrators for Multibody Dynamics", Technical Report 47, Department of Mathematics, University of Iowa, August 1993.

[37] L. R. Petzold and F. A. Potra, "ODAE methods for the numerical solution of Euler-Lagrange equations", Applied Numerical Mathematics 10 (1992), 397–413.

[38] F. A. Potra, "Implementation of Linear Multistep Methods for Solving Constrained Equations of Motion", SIAM Journal of Numerical Analysis, 30:3, June 1993, 774–789.

[39] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics (SIGGRAPH 87)*, pages 25–34. ACM, July 1987.

[40] W. Schiehlen (Editor), *Advanced Multibody System Dynamics, Simulation and Software Tools*, Kluwer Academic Publishers, London, 1993.

[41] A. J. Stewart and J. F. Cremer, "Algorithmic Control of Walking", *Proceedings of the 1989 International Conference on Robotics and Automation*, May, 1989, 1598–1603.

[42] G. Sun and P. Van Vleet and G. Vaněček, Jr. "Proxima, a Polyhedral Distance and Classification Support in C++", Department of Computer Science, Purdue University, Technical Report, CER–92–41, November 1992.

[43] F.F. Tsai and E.J. Haug, "Real-time multibody system dynamic simualtion, Part I — modified recusive formulation and topological analysis," *Mechanics of Structures and Machines*, 19:1, 1991.

[44] G. Vaněček, Jr., *ProtoSolid: An inside look*, Purdue University, Department of Computer Science, CER–89–26, November 1989.

[45] G. Vaněček, Jr., "Brep-Index: A Multi-dimensional Space Partitioning Tree," *First ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAD Applications*, Austin Texas, 35–44, June 1991.

[46] G. Vaněček, Jr., "A Data Structure for Analyzing Collisions of Moving Objects," *IEEE Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, Kailua-Kona Hawaii, Vol I:671–680, January 1991.

[47] H. Weghorst, G. Hooper, and D. P. Greenberg. "Improved Computational Methods for Ray Tracing," *ACM Trans. Graphics*, 3(1):52–69, 1984.

[48] J. Yen. "Constrained equations of motion in multibody dynamics as ODEs on manifolds," *SIAM Journal of Numerical Analysis*, 30:2, 1993, 553–568.

[49] J. Yen and C-C. Chou. "Automatic generation and numerical integration of differential-algebraic equations of multibody dynamics," *Computer Methods in Applied Mechanics and Engineering*, 104, 1993, pages 317–331.