Isaac: Building Simulations for Virtual Environments

James F. Cremer
University of Iowa
Computer Science Department, University of Iowa, Iowa City, IA, 52242 USA. email: cremer@cs.uiowa.edu

George Vaněček
Purdue University
Computer Science Department, Purdue University, West Lafayette, IN, 47907 USA. email: vanecek@cs.purdue.edu

Abstract

This paper describes the architecture and initial implementation of the *Isaac* system. Our general research goal is to develop simulation support for virtual environments. Existing virtual environments are often graphically rich, but behaviorally impoverished. On the other hand, existing physical system simulation software is not well-suited for use within virtual environments. The *Isaac* system is a distributed simulation server that integrates multibody dynamics, geometry, and control and is designed to support the needs of virtual environments.

Keywords

rigid-body dynamics, simulation, virtual environments, geometric modeling, contact, collisions, control.

1 INTRODUCTION

Existing virtual environments are often visually rich but behaviorally impoverished; users may be able to walk through geometrically complex worlds rendered using high-performance graphics hardware and software, but the worlds are typically populated with objects that do not behave as humans expect them to. Either users can only look at the objects, or the objects do not behave in physically satisfying ways — objects released from a hand don't fall, for example, or active non-user entities such as robots are purely scripted and not reactive. Such shortcomings 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 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.

The graphics, animation, and virtual environment communities have shown great 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 has been carried out. However, existing simulation tools were not designed specifically to support the requirements of virtual environments and, in fact, do not support them well. Dynamics simulation systems from mechanical engineering — e.g. DADS (Haug, 1989), ADAMS (Orlandea, 1987) — support analysis of mechanisms in a standard paradigm: formulate motion equations and kinematic constraints, and then numerically integrate them over time. 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 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, many are not technically sophisticated or 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).

In this paper, we introduce the architecture of *Isaac*, a distributed simulation server designed to provide efficient, robust, and flexible physically-based simulation within virtual environments. It is designed to support simulation of complex physical systems at interactive rates, to efficiently and robustly handle collision and contact phenomena, and to provide powerful, clean mechanisms for motion and scenario control. It will be possible to simulate virtual worlds populated with autonomous creatures under various modes of control. The control could range from simple kinematics-based scripting to semi-autonomous behavior and scenario control.

In the next section, we introduce the basic components of *Isaac*. Section 3 presents some historical context for *Isaac*, describing its roots in our earlier work on dynamics simulation and geometric modeling. Section 4 presents the *Isaac* system architecture, with sections covering the simulation core, dynamics, geometry, control, and distributed computation. The status of the *Isaac*, including the first implementation, is presented in Section 5.

2 ISAAC OVERVIEW

Designed to provide efficient, robust and flexible simulation support for virtual environments, *Isaac* comprises five basic components:

- a <u>simulation core</u> that contains state-of-the-art numerical methods and efficiently and robustly handles *on-line constraint changes*. In virtual environments collisions occur, contact relationships change, and motor control programs or high-level plans change state. In *Isaac*, these correspond to constraint changes in the underlying equations.
- a <u>dynamics module</u> 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.

Isaac integrates three components crucial to virtual environments—dynamics, geometry, and control. To ensure scalability and efficiency, it is designed for distributed computation.

3 BACKGROUND — *Isaac*'S FORERUNNERS

Our *Newton*(Cremer, 1989; Cremer and Stewart, 1989) system was one of the first attempts to integrate geometry and control with dynamics. One of the driving problems was the design of multifingered dextrous robot manipulators. 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. It was successful as an test for collision, contact, and control research. However, the contact and collisions module never reached an acceptably robust and efficient level, primarily because the interplay between dynamics and geometry had not been carefully studied before the development of *Newton* and was not well understood.

Until 1989, *Newton* dealt with geometry only as parameterized primitives. Later, Vaněček integrated the *ProtoSolid* polyhedral gemoetric modeling system with *Newton*, providing the system a broad range of convex and nonconvex polyhedral objects. The integration was difficult because *ProtoSolid* was originally designed to support mechanical design, not dynamics simulation. Specialized support for collision detection had to be added. This was accomplished in part by using Binary Space Partition (BSP) Trees (Thibault and Naylor, 1987). With the BSP tree support *Newton* could perform simulations with any polyhedra, but only for simple contacts. Complex contact models such as the simulation of the tumbling rings, shown in Figure 1, failed due to insufficient contact information obtainable from the BSP support. Consequently, Vaněček generalized the trees to multi-dimensional structures and later to the Brep-Index, and is incorporating this in a system called Proxima (Sun, Van Vleet, and Vaněček, 1992).

The control portion of *Newton* was its most successful component. Paradigms for *programming* the control of high-degree-of-freedom mechanisms were developed and applied to 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 (Cremer and Kearney, 1994). At Iowa, Hansen develop a programming framework that enabled users to create complex simulations involving multiple interacting robots from clear, concise, control programs (Hansen, 1993). At Purdue, Bouma was able to program control a four-legged walking robot in *Newton*(see Figure 2). At Cornell, Pai and others programmed bipeds and hopping machines for tasks including standing-up, sitting, walking, jumping and riding a bicycle (Pai, 1991; Stewart and Cremer, 1992; Kearney, Hansen, and Cremer, 1993).

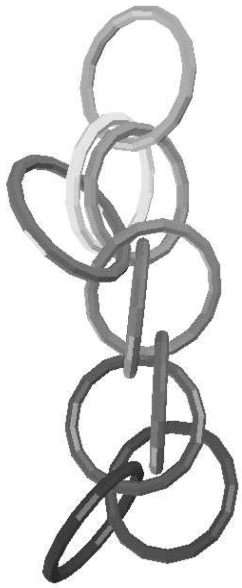

Figure 1 *Newton* simulation of tumbling rings. This idea was obtained from an article in "Mathematical Games", Scientific American, 1965.

4 *Isaac* ARCHITECTURE

The basic *Isaac* architecture is shown in Figure 3. 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.

4.1 Simulation core

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*'s simulation core has two major goals; namely,

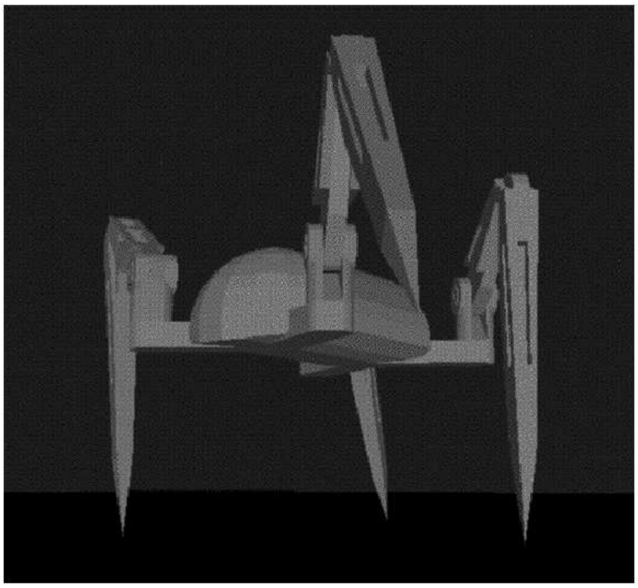

Figure 2 Controlling GUMBY, a four-legged simulated walking robot.

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

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. For example, for some problems standard DAE solvers like MEXX (Lubich et al., 1992) 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 4.2 will be necessary. For efficiency purposes, the constraint programming interface routines will each have a number of implementations based on the various solvers. If instead, we had a generic set 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* by specifying how an event is to be *detected* and how it is to be *resolved.* Event detection may correspond to a

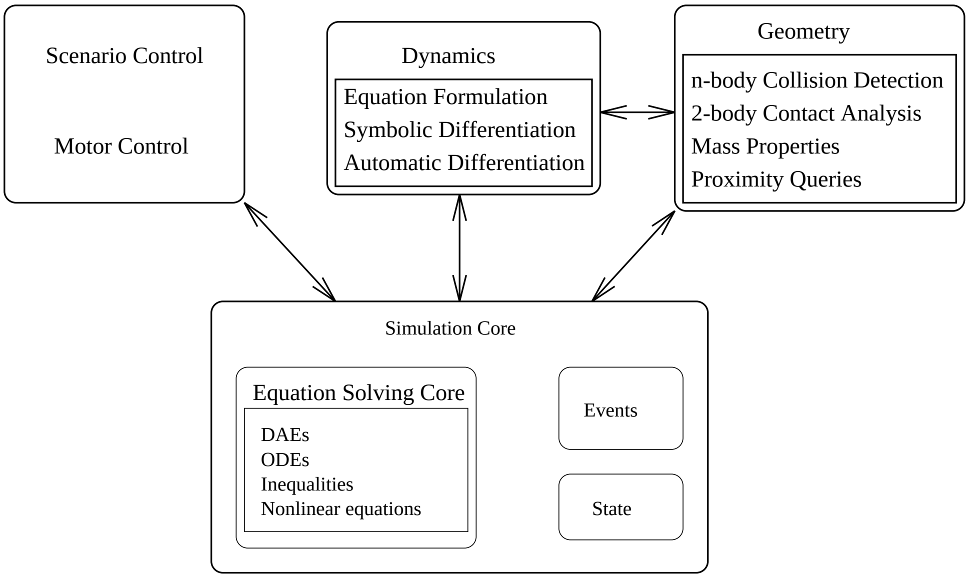

Figure 3 The *Isaac* architecture.

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 event handling; event occurrence time must be *isolated* efficiently, and continuation of the simulation after an event must be done with maximal efficiency and accuracy.

4.2 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) corresponding 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 uses a maximal set of Cartesian coordinates, in the style of Haug/DADS (Haug, 1989) and Cremer/Newton (Cremer, 1989). 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 (Tsai and Haug, 1991; Bae and Haug 1987), will be introduced.

When contact constraints, called *temporary constraints*, are present, the dynamics module interacts with the geometry module to formulate the appropriate set of equations*. As outlined below, a contact

*Throughout the paper, we use the term *equations* to include inequalities as well as true equations.

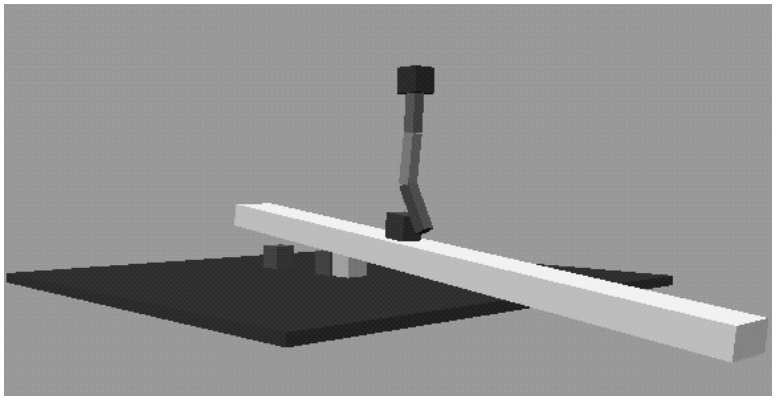

Figure 4 A simple simulation involving collisions and sliding contact.

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. As an example, Figure 4 shows a set of blocks in temporary contact.

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 pulling up on the block such that it breaks contact by lifting off the face. This is a dynamics event — it arises because the non-compressive 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 instant 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 signaled and the contact analysis routines analyze the situation and update the equations with new contact constraints.

To most efficiently support our model of dynamics-geometry integration, novel numerical integration techniques are 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. Currently, Cremer and F. Potra are are working to develop to develop a new DAE integrator that is designed especially to efficiently and robustly handle changing constraint sets that include inequalities.

4.3 Geometry

Geometric support for simulation in virtual environments must include:

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 prediction,
4. fast two-body collision detection, and
5. fast and robust two-body contact analysis.

Representing the Geometric Environment

Currently, to limit complexity, we assume that environments consist of nondeformable objects that do not interpenetrate. We partition the objects into two categories: movable objects and fixed, immovable objects (e.g. the ground or walls). Objects that move are currently modeled using planar polyhedra. We plan to extend the representations to include free-form surfaces. Fixed objects can be modeled using relatively large, oriented lamina (i.e., surfaces) and not closed volumes. In terms of the dynamics, the objects that do not move need not have their mass properties computed or motion equations formulated; when convenient, all fixed objects can be combined geometrically into a single complex object.

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 inertia matrix which encodes the moments of inertia of the object. Since the objects are assumed rigid, the inertia matrix does not change during a simulation and can be precomputed. This is a well-understood problem for which efficient boundary-based algorithms exist.

Collision Detection

Isaac relies on automatic detection of collisions between objects. 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

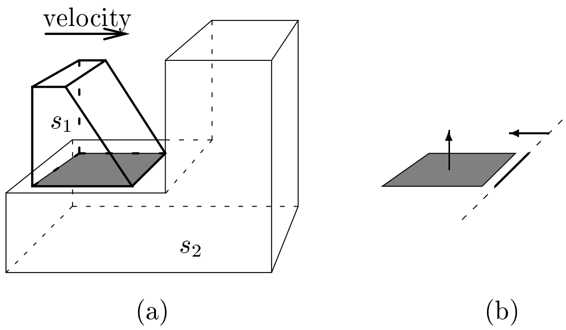

Figure 5 *An example where a two objects in temporary-contact will also collide. (a) show the objects at time t, and (b) shows the two regions at time $t + \Delta t$, one for the temporary-contact, the other for the collision.*

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.

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, one may approximate objects that are far apart by their convex hull and then check their proximity using the fastest known algorithm for convex object, the Lin and Canny's algorithm (Lin and Canny, 1991). 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 (Lin, Manocha, and Canny, 1993). A similar extension was proposed by Dworkin and Zeltzer at MIT; it uses simple time-parameterized object trajectories to predict possible intersections (Dworkin and Zeltzer, 1993). 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 time interval is small, the trajectories of objects are known a priori and that objects are relatively far away from each other. These assumptions are not always reasonable.

In our case, however, we use a totally different approach mainly due to the assumption that objects can be of any shape and that they can be in prolonged contact. We are developing an n-body collision detection algorithm that integrates three components: a Locally-resolvable Boundary Representation (LRBep) (Gonzalez-Ochoa and Vaněček, 1994), an Extended Binary Space Partitioning (EBSP) tree, and lazy evaluation. LRB-reps are basically a generalization of bounding-volumes. The faces of a B-rep are locally abstracted to form a single multi-resolution structure that can support a locally resolvable wrapper. A wrapper is a subset of the LRB-rep's faces which enclose the original object but which can be locally refined to higher levels of detail. Each object can thus be approximated by a wrapper. As two objects come close, their wrappers may intersect. However, these wrapper can be locally refined at the intersection to expose more detail. To attain the n-body collision check, the wrappers of all objects are

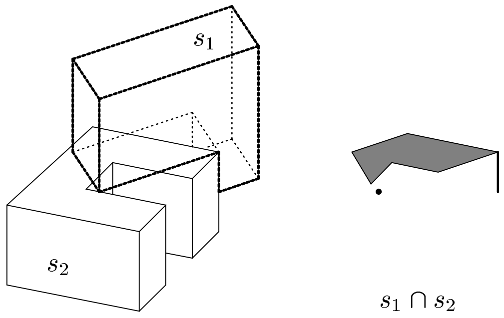

Figure 6 *Two objects in contact, and their three contact regions resulting from a set-theoretic intersection.*

inserted into a global BSP tree. The insertion is only partial in that a wrapper is not completely processed by the BSP tree. Only the parts that conflict with other wrappers are processed and only at local regions of intersections are the wrappers refined to higher levels of detail.

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 (Bouma and Vanecek, 1993). 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. For large N, the cost can be prohibitive. To speed up this analysis, Vaněček has developed two techniques: Brep-index (Vaněček, 1991) and back-face culling (Vaněček, 1994). Basically, the analysis begins by culling the vertices, edges and faces that are know a priori to be moving away from the other object and therefore cannot be 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.

The Brep-Index

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 the binary-space partition (BSP) tree. The MSP structure allows for a fast search that quickly converges to the region containing the query point, line segment or polygon. To gain the benefit of both the efficient spatial 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 (Vaněček, 1991b).

Back-Face Culling

In order to find contact regions, we can classify all the faces, edges and vertices of one of the objects against the Brep-index of the other object. Although this reduces the cost of classification from quadratic to subquadratic, we can reduce the cost even more. Vaněček has applied the well-known back-face culling idea of computer graphics to that of the contact analysis, basing decisions relative velocity instead of on view direction. A face may be culled (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. Combining the back-face culling technique with the brep-index yields a very efficient technique for detecting collisions for objects in close proximity and for determining contact regions.

Proxima

Proxima is a set of C++ routines provided as a library, and intended to provide the geometric support described in this section (Sun, Van Vleet, and Vaněček, 1992). The primary representation for objects is the BRep, with the MSP tree and the Brep-index as secondary representations. Through these representations, *Proxima* provides a wealth of low-level geometric operations to query the boundary, classify entities, and to obtain mass properties. Initial implementations of the contact analysis and collision detection components of *Proxima* are complete, but significant further development is required.

On the Geometric Complexity

On first inspection, it may appear that the geometric support is unnecessarily 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 choices for the data structures and algorithms, we feel that we've chosen the best of the current state of the art.

4.4 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.

Isaac is designed to support both motion control and scenario control. As described in the Section 4, 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 (Kearney, Hansen, and Cremer, 1993; Hansen, 1993; Hansen, Kearney, and Cremer, 1994) on control for mechanical simulation and on related work by others (Harel, 1987; Brooks, 1989; Reynolds, 1987). The framework is based on a notion of concurrent, hierarchical state machines that is currently being developed by Cremer and Kearney for behavior modeling and scenario control in Iowa Driving Simulator (Cremer and Kearney, 1994; Ahmad et al., 1994). We also intend to integrate our work on control some of the Bates' work (Bates, 1992; Kelso, Weyrauch, and Bates, 1993) on "believable agents".

4.5 Distributed computation with *Isaac*

The geometry, dynamics, control, and simulation core components of *Isaac* are being 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* consists of a set of *Isaac* simulation-server processes managed by a *task management* process. Each *Isaac* process is a self-contained simulation server; the entire computation *could* be done within any single process. However, it is the combined responsibility of server processes and the task manager to distribute computations across multiple *Isaac* processes. For example, at a fairly simple level, there can be one *Isaac* process for each independent (e.g. kinematically independent) mechanism. Each process acquires 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 communication is reasonable within *Isaac* because the anticipated maximum number of entities is relatively small (i.e. hundreds of moving objects).

5 IMPLEMENTATION STATUS

Isaac is being developed jointly by the University of Iowa and Purdue University. Iowa is responsible for the design of the overall system architecture, and for the dynamics and control components of the system. Purdue is responsible for addressing the substantial geometric demands of *Isaac*.

Isaac ran its first simulations in July 1994. The first version was able to simulate multiple fairly simple articulated objects (e.g. pendulums) in real-time (30 frames/second or better) on a network of workstations. Individual workstations simulate one or more mechanisms, depending on their complexity.

We developed a deterministic time, distributed virtual environment system that allowed users to sit at graphics workstations and interactively examine the simulation. Visualization clients can run on all workstations in a local-area network, each displaying a unique view into the synchronized environment.

Isaac is being developed in C++ on Silicon Graphics workstations. Two visualization clients have been developed — one using openGL and a simpler X windows-only version. A tool that allows users to create complex mechanism and manipulate them into desired initial configurations is being developed using SGI's Open Inventor. Future versions of *Isaac* will be publically available.

6 CONCLUDING REMARKS

The development of high-performance computer graphics hardware and software, head-mounted displays, and a range of position tracking, haptic, and multi-degree of freedom interaction devices, has enabled the creation of visually rich interactive virtual environments. One of the most important, but least developed, facets of virtual environments research is the support for *behavior*. To provide a realistic, satisfying experience, users must be able to interact with objects they encounter in virtual environments, and have those objects behave according to a physically-based model. Users must be able to pick up, carry, push, throw, swing various things they encounter in then environment. Furthermore, a successful environment oftens requires more complex, active, semi-autonomous entities — objects, such as robots or humanoids, that have a substantial control component integrated with the physically-based model.

Our system, *Isaac*, is being designed to address the simulation needs of virtual environments. By creating a distributed simulation server that integrates efficient dynamics with geometric computation and control programming facilities, *Isaac* will make it possible to populate virtual environments with complex objects exhibiting believable and interesting physically-based behavior.

7 ACKNOWLEDGEMENTS

This work is supported by Office of Naval Research grant N00014-94-1-0576. Iowa's Michael Booth and Paul Henning developed much of the first *Isaac* implementation.

REFERENCES

Ahmad, O., Cremer, J., Hansen, S., Kearney, J. and Willemsen, P. (1994) Hierarchical, concurrent state machines for behavior modeling and scenario control. In *Proceedings of the 1994 Conference on AI, Simulation, and Planning in High Autonomy Systems*, Gainesville, December, pp. 36–42.

Bae, D. S. and Haug, E. J. (1987) A recursive formulation for constrained mechanical systems, part II — closed loop. *Mechanics of Structures and Machines*, **15**(4).

Joseph Bates. (1992) Virtual reality, art, and entertainment. *PRESENCE: Teleoperators and Virtual Environments*, **1**(1), pp. 133–138.

Bouma, W. and Vaněček, G. (1993) Modeling contacts in a physically based simulation. In *Proceedings of 2nd ACM Symposium on Solid Modeling and Applications*, Montreal, May 1993, pp. 409–418. (revised version to appear in Computer Aided Design).

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

Cremer, J. (1989) *An Architecture for General Purpose Physical System Simulation — Integrating Geometry, Dynamics, and Control*. PhD thesis, Cornell University, May 1989.

Cremer, J. and Kearney, J. (1994) Scenario authoring for virtual environments. In *Proceedings of the IMAGE VII Conference*, Tucson, AZ, June, pp. 141–149.

Cremer, J. and Stewart, A. J. (1989) The architecture of Newton, a general-purpose dynamics simulator. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, May, pp. 1806–1811.

Dworkin, P. and Zeltzer, D. (1993) A new model for efficient dynamic simulation. In *Proceedings of the 4th Eurographics Workshop on Animation and Simulation*, Barcelona, September 1993, pp. 135–147.

Gonzalez-Ochoa, C. and Vaněček, G. (1994) Locally resolvable b-reps (revised january 1995). Technical Report TR-94-076, Dept. of Computer Science, Purdue University, November 1994.

Hansen, S. (1993) *Conceptual Control Programming for Physical System Simulation*. PhD thesis, Computer Science Department, University of Iowa, May 1993.

Hansen, S., Kearney, J. and Cremer, J. (1994) Motion control through communicating, hierarchical state machines. In *Proceedings of the 5th Eurographics Workshop on Animation and Simulation*, Oslo, September 1994.

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

Haug, E. (1989) *Computer Aided Kinematics and Dynamics of Mechanical Systems, Volume I: Basic Methods*, Allyn and Bacon.

Kearney, J., Hansen, S. and Cremer, J. (1993) Programming mechanical simulations. *The Journal of Visualization and Computer Animation*, **4**(2), April-June 1993, pp. 113–129.

Kelso, M., Weyhrauch, P. and Bates, J. (1993) Dramatic presence. *PRESENCE: Teleoperators and Virtual Environments*, **2**(1).

Lin, M. and Canny, J. (1991) Efficient algorithms for incremental distance computation. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*.

Lin, M., Manocha, D. and Canny, J. (1993) Fast collision detection between geometric models. Technical Report TR93-004, Department of Computer Science, University of North Carolina at Chapel Hill, January 1993.

Lubich, Ch., Nowak, U., Pöle, U. and Engstler, Ch. (1992) Mexx — numerical software for the integration of constrained mechanical systems. Technical Report Technical Report SC-92-12, Konrad-Zuse-Zentrum für Informationstechnik, Berlin.

Orlandea, N. (1987) Adams – theory and practice. *Vehicle Systems Dynamics*, **16**, pp. 121–166.

Pai, D. (1991) Least constraint: A framework for the control of complex mechanical systems. In *Proceedings of the American Control Conference*, Boston, MA, American Automatic Control Council, pp. 1615–1621.

Reynolds, C. (1987) Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics (SIGGRAPH 1987)*, ACM, July, pp. 25–34.

Stewart, A. J. and Cremer, J. (1992) Animation of 3d human locomotion: Climbing stairs and descending stairs. In *Proceedings of the 3rd Eurographics Workshop on Animation and Simulation*, Cambridge, England, September 1992.

Sun, G., Van Vleet, P. and Vaněček, G. (1992) Proxima, a polyhedral distance and classification support in C++. Technical Report CER-92-41, Dept. of Computer Science, Purdue University, November 1992.

Thibault, W. and Naylor, B. (1987) Set operations on polyhedra using binary space partitioning trees. In *Computer Graphics (SIGGRAPH 1987)*, ACM, July, pp. 153–162.

Tsai, F. and Haug, E. (1991) Real-time multibody system dynamic simulation, part I — modified recusive formulation and topological analysis. *Mechanics of Structures and Machines*, **19**(1).

Vaněček, G. (1991) Brep-index: A multidimensional space partitioning tree (revised). *International Journal of Computational Geometry and Applications*, **1**(3), September 1991, pp. 243–262.

Vaněček, G. (1991b) Brep-index: A multi-dimensional space partitioning tree. In *Proceedings of 1st ACM Symposium on Solid Modeling and Applications*, Austin, June 1991, pp. 35–44.

Vaněček, G. (1994) Back-face culling applied to collision detection of polyhedra. *Journal of Visualization and Computer Animation*, **5**(1), January 1994, pp. 55–63.

AUTHOR BIOGRAPHIES

James F. Cremer is an Assistant Professor of Computer Science at the University of Iowa. He received his Ph.D. from Cornell University in 1989. He remained at Cornell as a Research Associate from 1989–1992 and led research and development of the Newton dynamics simulator and, with Rick Palmer, initiated the SimLab project. Since moving to Iowa in 1992, he has been active in research in environments for constructing simulators (SimLab and related projects), scenario control for real-time vehicle simulation (with Iowa's Center for Computer-Aided Design and the Iowa Driving Simulator), and techniques for efficiently handling constraint changes related to contact, collision, and control in simulation-based virtual environments (project Isaac).

George Vanecek is an Assistant Professor in the Department of Computer Sciences at Purdue University. His work includes geometric modeling, computer graphics, animation, simulation, virtual environments, and recently, enterprise integration. In 1992, trying to solve the collision problem, he formulated a representation of polyhedra that unified a spatial representation, a multi-dimensional space partition trees, and a boundary representation, called the Brep-Index. The work later led to a model of contact in his work on contact-analysis of objects. Currently he is working on the geometric support for simulations of complex virtual environments and on virtual enterprise integration on the World-Wide Web. Vanecek received his Ph.D. from the Department of Computer Science at University of Maryland, College Park.