Techniques for Virtual Environments

SIMULATION AND SCENARIO SUPPORT FOR VIRTUAL ENVIRONMENTS

JAMES CREMER,† JOSEPH KEARNEY* and HYEONGSEOK KO†
1Computer Science Department, University of Iowa, Iowa City, IA 52242, U.S.A. and
2 School of Electrical Engineering, Seoul National University, Seoul, Korea
e-mail: cremer@cs.uiowa.edu

Abstract—Over the past few years, we have developed several research projects focusing on critical support technologies for virtual environments. In this paper, we outline three aspects of our effort: simulation support, scenario authoring, and real-time animation of human locomotion. Our work on physical simulation combines dynamics and collision/contact analysis to provide physical realism to virtual environments. Our work on scenario authoring supports both modeling reactive semiautonomous behavior of complex entities, and orchestration of groups of such entities to satisfy the goals or intentions of an experimenter/author. This effort evolved from research on the Iowa Driving Simulator, a large-scale virtual driving environment. Our work on real-time animation of human locomotion concentrates on methods to generate realistic human walking motions from simple positional control parameters. Locomotion can be along an arbitrary curved path. The resulting animation switches among five walking modes—walking, running, lateral stepping, backward stepping, or turning around—based on the current velocity and heading. We present demonstrations of our work in the three areas and outline the directions of future research. Copyright © 1996 Elsevier Science Ltd

1. INTRODUCTION

Over the past few years, we have developed several research projects that are creating critical support technologies for virtual environments. This work was inspired by our belief that rapid progress in image generation and user-interface hardware for virtual reality has not been paralleled by developments in real-time interactive simulation and scenario software. Many potential virtual environments applications, especially those involving multiple interacting or collaborating users, cannot be realized without substantial developments in the supporting software.

In this paper, we outline three aspects of our effort to provide supporting software technology for real-time interactive virtual environments:

1. Simulation support: virtual environment applications will increasingly call for worlds in which users can interact with the objects they encounter in natural, expected ways. Virtual objects should behave in physically reasonable ways—they should conform to physical laws and should respond appropriately when, for instance, users touch, push, or throw them.

2. Scenario authoring: as virtual environments become more complex, the problem of "programming the action" within them looms larger. VE designers want certain things to happen in the environment. They may first want autonomous agents that exhibit complex interesting behaviors. But, usually, that's not all—they also want certain specific things or kinds of things to happen in the environment, perhaps in some specified order or in response to specified events. Designers want to be able to author scenarios—to inconspicuously orchestrate or direct the action in a VE (while not inhibiting the VE user's sense of a free, unconstrained environment).

3. Real-time animation of human locomotion: multi-user virtual environments have a clear need for modeling and animating humans. In such environments, some of the animated humans may represent actual human users of the VE, while others may represent computer-controlled autonomous agents. In either case, real-time natural-looking human animation is important.

In the following three sections, we briefly describe our work in each of the areas.

2. SIMULATION SUPPORT

The Isaac project [1] is developing a new simulation architecture specifically designed to provide efficient, robust, and flexible physically-based simulation within virtual environments. It is designed to support simulation of multibody systems in real-time, to efficiently and robustly handle collision and contact phenomena, and to provide powerful, clean mechanisms for motion and scenario control. Eventually, 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 semiautonomous behavior and scenario control. The project is an outgrowth of the Newton project [2], with emphasis on robust methods and real-time support of virtual environment applications.

As a software system, Isaac is a distributed simulation server designed and organized to provide efficient, robust, and flexible physically-based simulation within virtual environments. It comprises five key modules:

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

2. 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.

3. A geometry module that efficiently and robustly supports n-body first contact 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 collision avoidance planning.

4. 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 for coordinating high-degree-of-freedom mechanisms such as anthropomorphic robots) as well as scenario and behavioral control (including coordination of multiple agents, planning and control of high-level agent behavior).

5. 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.

2.1. Discussion

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 (DADS, ADAMS, RASNA, etc.) support analysis of mechanisms and machines in a standard paradigm: formulate some equations, and 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 efficient on-line n-body collision detection and two-body contact analysis. 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 relatively low. 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).

In contrast, Isaac is being developed specifically to address the problems of real-time interactive simulation in VEs. Thus, Isaac servers may be “plugged into” virtual environments to provide physical simulation for entities in the modeled environment. Building Isaac requires a number of significant technical developments. First, the real-time requirements of virtual environments necessitate development of a new software architecture that supports the tight integration of dynamics simulation, geometric computation, and control. Second, new geometric modeling techniques must be developed that support fast, incremental, and robust collision detection and contact region classification. In virtual environments, collisions and contact must be considered usual and expected occurrences. Third, to enable real-time simulation that includes collision and contact phenomena, we must develop new numerical integration techniques designed specifically to handle systems of differential-algebraic equations subject to discontinuities and inequality constraints. Figure 1 shows a frame from a simple simulation produced with the existing Isaac implementation.

Geometric support for Isaac is provided by the Proxima system [3]. Proxima is a library of C++ routines providing a family of geometric representations and algorithms required for simulation. Representations include a non-manifold boundary representation, MSP (multi-dimensional space partitioning) trees and the Brep-index [4], and a new locally resolvable non-manifold boundary representation, LR-Brep [5], that is especially well-suited for efficient collision detection. Through these representations, Proxima provides a wealth of low-level geometric operations to query boundaries, classify entities, obtain mass properties, detect collisions and compute contact regions.

By providing dynamics, Isaac will strongly support the “passive” behavior aspects of virtual environments. But, as we argue in the scenario authoring section (Section 3), this is only a partial step toward richer virtual environments. It is vitally important that we also develop tools that enable controlling or programming “active” behaviors of complex objects. Thus, Isaac is also designed to provide clean interfaces for control. It supports scenario control as described in Section 3, as well as lower-level motion control, which 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. The control subsystem interacts 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, programs will be specified in a framework based on the hierarchical, concurrent state machines described in Section 3.2.

2.1.1. Status. The current implementation of Isaac is able to simulate multiple fairly simple articulated objects at 30 frames per second on a network of workstations. Individual workstations simulate one or more mechanisms, depending on their complexity. Collision detection and contact analysis using Proxima is implemented, though not all situations are currently handled and, for some cases, real-time response has not yet been achieved. New techniques for simulating contact dynamics involving curved surfaces have been developed and implemented (see [6]). We developed a deterministic-time, distributed virtual environment system that allowed users to sit at graphics workstations and interactively examine and interact with the simulation. Visualization clients can run on all workstations in a local-area network, each presenting a unique view into the synchronized environment.

3. SCENARIO AUTHORING

Our work in scenario authoring has grown out of projects in two areas—creating experiment-specific traffic for the Iowa Driving Simulator (IDS) and programming control of high-degree-of-freedom mechanisms, such as virtual humans and animals. In these domains, we discovered the need for techniques that support both (1) modeling reactive semiautonomous behavior of complex entities and (2) orchestration of groups of such entities to satisfy the goals or intentions of an experimenter/author. The IDS is introduced in the following subsection, and the approaches taken to achieve the above two goals are explained in the subsequent sections.

3.1. Overview of the Iowa Driving Simulator

Much of our current scenario authoring effort is focused on the Iowa Driving Simulator. Figure 2 shows the motion platform (motion base) of the IDS. The IDS is a high-fidelity operator-in-the-loop ground vehicle simulator that incorporates a motion platform, force feedback and control loading, high-quality visuals, sound, state-of-the-art real-time multibody dynamics, and scenario control. A Ford Taurus cab is mounted inside a dome on top of the motion platform. High resolution, textured graphics are projected on screens on the dome walls—the forward field of view is 191° x 45° and the rear field of view is 64° x 35°. See Kuhl [7] for a detailed description of the IDS.

The IDS is an operational, useful, rich virtual environment; it provides a safe, virtual environment for a variety driving experiments, including the influence of drugs, disease, and disabilities on driving performance; the effectiveness of computer-assisted

---

† Other cabs can be installed in the dome—IDS uses HMMWV and Saturn cabs for some of its projects.
driving aids such as collision warning devices and intelligent information systems, and the response of humans to critical events such as sudden, unexpected braking by leading vehicles. One of the critical problems in IDS development is the time and expense involved in creating the different scenarios and database scenes required by the various experiments.

3.2. Scenario control and authoring using HCSM

We decompose the scenario authoring problem into two components: (1) modeling the basic, reactive behavior of the dynamic entities and (2) modeling scenarios using coordination mechanisms that organize the behaviors of entities to create situations and critical events.

3.2.1. Behavior modeling. In the first generation IDS software, vehicle behaviors were controlled with complex, one-level state machines. These state machines modeled driving on an open road, following, passing, and intersection behavior. Using state machines, the scenario control subsystem could generate ambient traffic that had a natural feel, and in which familiar phenomena such as jams emerged. However, we found these programs difficult to modify and debug. The source of the problem was that the single-level state machines tended to become large and densely interconnected. As a consequence, small changes tended to propagate throughout the state machine. In addition, the single-level state machines provided no mechanism for abstraction or encapsulation. This made it difficult to independently test and refine what appeared to be logically separate behaviors. It also inhibited reuse and recombination of component behaviors. Lastly, the single-minded focus and sequential logic of non-concurrent state machines made it difficult to capture the natural contention among the multiple demands on a driver to simultaneously follow the road, obey the speed limit, avoid collisions, and monitor traffic signals.

The second generation of scenario control uses hierarchical concurrent state machines (HCSM) to model object behaviors. As detailed in [8], HCSM extends the state machine model to include hierarchies of concurrent state machines. An HCSM is composed of one or more concurrently executing sub-state machines. The introduction of a hierarchical structure supports abstraction by allowing states and transitions representing coherent activities to be grouped. The concurrency of HCSM addresses the problem of attending to multiple constraints or goals during driving. At each instant, an HCSM integrates all the relevant information from sub-state machines to determine how to adjust control parameters.

Haral's Statechart formalism [9], Reynolds' work on flocking behaviors [10], and Brooks' "subsumption" architecture [11] each significantly influenced the development of HCSM.

3.2.2. Modeling situations and scenarios. Experimenters studying human behavior are interested in using virtual environments to test the responses of subjects to specific circumstances. For example, investigators studying driving safety would like to expose IDS subjects to specific crash threats. To elicit natural responses from subjects, the situations encountered must seem uncontrived; threatening events such as lane encroachment, sudden deceleration, and intersection departures, must occur in the natural course of the simulation without arousing suspicions that would permit subjects to prematurely anticipate critical events. Object behaviors must be plausible
and consistent with the subject's experience—vehicles cannot suddenly appear out of thin air and traffic signals must appropriately cycle through their stages. The challenge for scenario control is to create complex scenarios that meet the experimenter's need for replicability while maintaining diversity, reactivity, and realism in object behaviors.

We are creating tools that will allow adaptive direction of object behaviors during simulations. Our goal is to create controlled experimental situations and event sequences that blend smoothly with a background of autonomous, reactive behaviors. Coordination is accomplished by behavior controller HCSMs that monitor the simulation's progress and manipulate the environment by sending instructions to other objects.

3.2.3. Mechanisms for interaction. To provide means for directing the actions of scenario objects, HCSM state machines include communication interfaces that allow them to exchange control information. Each state machine has a software control panel consisting of 'buttons' and 'dials' whose values influence the execution of the object's state machines. An active object or the experimenter can influence another object's behavior by sending messages indicating desired settings of instruments on its control panel. It may happen that multiple state machines simultaneously request that the same dial be set to different values. The state machine receiving the conflicting requests is responsible for arbitrating among the requests.

3.2.4. Coordinating processes. We use behavior controllers to direct object behaviors to create critical events. For convenience, we classify behavior controllers according to how they are activated and how they interact with other objects. A 'trigger' is a behavior controller that is placed at a specific location on a roadway and is activated when a vehicle drives over it. When activated, a trigger pushes a button on another object instructing it to change its behavior. For example, a trigger can be used to initiate the motion of a vehicle on the shoulder of a highway as the subject’s vehicle approaches it.

A trigger affects predetermined objects. When we cannot predict which particular objects must play roles in a situation until run-time, we use beacon behavior controllers. A 'beacon' radiates messages to nearby objects; it can be placed at a specific location or it can be attached to a moving object. For example, a beacon can be attached to the subject vehicle and at the appropriate time instruct the vehicles in front of the subject to accelerate or change lane in order to create a clear path for the subject.

Beacons can be used to coordinate the actions of objects. Consider an experiment in which we want to test a subject's response to a vehicle running a red light as the subject approaches an intersection. It is ineffective to pre-select the offending vehicle, subjects drive at different rates and will arrive at the intersection at different times, making it difficult to guarantee that a particular car will be in the right place at the right time. Instead, a beacon can be used to watch for the subject vehicle and conscript an appropriate scenario vehicle to run the light. The beacon may help set conditions for the event by sending a repelling directive to nearby vehicles to create a pocket of clear roadway around the subject in preparation for the event. Thus, we can present a consistent set of circumstances across a series of trials without sacrificing the dynamism and reactivity essential to creating a believable scenario.

Figure 3 shows a snapshot of the visualization software used for designing and debugging vehicle behaviors and scenarios for the IDS. Debugging behaviors and scenarios on the full simulator is quite costly. The scenario visualizer, together with HCSM programming and debugging tools, allows cost-effective evaluation and refinement of behaviors and scenarios on a standard graphics workstation.

4. REAL-TIME ANIMATION OF HUMAN LOCOMOTION

Human locomotion occupies a large part of human activity. In order to create a realistic presence of other humans in virtual environments, we must find ways to control and animate the motions of human models. The source of the motion may be from direct interactions of human participants or may be from programmatically derived synthetic agents whose purpose is to create a scenario. In either case, it is important that the humans populating a virtual environment move from place to place in a natural way.

Generating a realistic animation of human locomotion is a difficult task. Flexible methods are needed to generate the range of everyday variations on normal rhythmic forward walking, including curved path walking, lateral stepping, backward stepping, turning around, ducking, running, stepping over, running to walking and walking to running transitions, and so on. Efficiency of the algorithm is important for real-time simulation.

Animation techniques such as key-framing and rote replay of captured motions are poorly suited to interactive environments. Much of the previous work on algorithmically generated locomotion is limited to straight-line or simple curved-path walking. These simple schemes are also not amenable to real-time, interactive use.

Our approach allows interactive real-time generation and control of human locomotion from simple inputs—desired body center position and facing direction over time. The approach is based on biomechanical measurements of straight-line walking for typical walkers. The motion is generalized to fit different body sizes and adjusted for any path curvature, turning, or steps [12–14]. The motion characteristics of the original gait are preserved during the generalization even if the step direction or step length is changed [12]. Therefore several different walking styles can be adopted merely by acquiring multiple sets of measurements. Figure 4 shows a frame from a curved path walk.
Our locomotion method is not limited to rhythmic walking. We can generate non-periodic steps in arbitrary directions such as lateral, backward, direction reversal, etc. Running and the transitions between running and walking are also included in our locomotion algorithm.

As a virtual human moves through an environment, she should modify her walking pattern to reflect environmental constraints. For example, a crouched walking posture might be required to avoid low hanging obstacles or to avoid being seen. The motor level of the locomotion algorithm should provide input for the pattern of stepping as well as path and direction information. To facilitate stylistic
variations, our methods include primary and attribute parameters that a higher-level component may specify. Primary parameters must be specified to generate a step; they provide the position of the next heel strike, the direction of the next foot, and a designation of the stepping foot (left or right). For the attribute parameters, default values are used to produce a normal motion. Higher level components can specify the values of attribute parameters to achieve a specific goal. Attribute parameters consist of the swing ankle height at its apex, the pelvis height at its apex, and other rotational and translational attributes of the torso and pelvis. For example, setting the swing ankle height to a greater value will allow the agent to step over a shallow obstacle.

Locomotion generation becomes even more difficult if we wish to simulate the effect of attached loads or external forces. If the animated motion is to be dynamically sound, we need to impose high-level constraints on the system; the overall motion should be dynamically balanced, and joint torques should be maintained within a moderate range imposed by the body strength. In our locomotion method, real-time inverse dynamics, balance and comfort control are performed so that the resulting motion is not only visually realistic but also dynamically sound. Our approach is to dynamically adjust motions to keep constraint violations at low level.

Our recent work [15] focuses on techniques for animating human locomotion in real-time from simple positional control parameters: the position of the body center and the facing direction. These input parameters are supplied on-line and can be supplied by a human using a VR device or by a program controlling the behavior of a synthetic agent.

Our locomotion methods can switch between five walking modes—walking, running, lateral stepping, backward stepping, or turning around. The appropriate mode is chosen on-line, based on recent input history. When a mode change is necessary, the system generates transitional motion that smoothly blends between the stepping styles.

4.1. Discussion

We have experimented with animating and controlling human locomotion through a number of interfaces direct user control through the use of Motif widgets, interactive control through a VR device consisting of a stationary bicycle instrumented with sensors to measure steering direction and pedal rotation, and HCSM programs implementing autonomous agents.

Our technique is broadly useful. It has been demonstrated in a reactive incremental path planner [16] and a virtual combat environment [17]. The path planner controlled an agent traveling through a terrain in which environment changed dynamically. The agent's path was incrementally computed, and other rotational and translational attributes of the torso and pelvis. For example, setting the swing ankle height to a greater value will allow the agent to step over a shallow obstacle.

Locomotion generation becomes even more difficult if we wish to simulate the effect of attached loads or external forces. If the animated motion is to be dynamically sound, we need to impose high-level constraints on the system; the overall motion should be dynamically balanced, and joint torques should be maintained within a moderate range imposed by the body strength. In our locomotion method, real-time inverse dynamics, balance and comfort control are performed so that the resulting motion is not only visually realistic but also dynamically sound. Our approach is to dynamically adjust motions to keep constraint violations at low level.

Our recent work [15] focuses on techniques for animating human locomotion in real-time from simple positional control parameters: the position of the body center and the facing direction. These input parameters are supplied on-line and can be supplied by a human using a VR device or by a program controlling the behavior of a synthetic agent.

Our locomotion methods can switch between five walking modes—walking, running, lateral stepping, backward stepping, or turning around. The appropriate mode is chosen on-line, based on recent input history. When a mode change is necessary, the system generates transitional motion that smoothly blends between the stepping styles.

4.1. Discussion

We have experimented with animating and controlling human locomotion through a number of interfaces direct user control through the use of Motif widgets, interactive control through a VR device consisting of a stationary bicycle instrumented with sensors to measure steering direction and pedal rotation, and HCSM programs implementing autonomous agents.

Our technique is broadly useful. It has been demonstrated in a reactive incremental path planner [16] and a virtual combat environment [17]. The path planner controlled an agent traveling through a terrain in which environment changed dynamically. The agent's path was incrementally computed, and other rotational and translational attributes of the torso and pelvis. For example, setting the swing ankle height to a greater value will allow the agent to step over a shallow obstacle.

Locomotion generation becomes even more difficult if we wish to simulate the effect of attached loads or external forces. If the animated motion is to be dynamically sound, we need to impose high-level constraints on the system; the overall motion should be dynamically balanced, and joint torques should be maintained within a moderate range imposed by the body strength. In our locomotion method, real-time inverse dynamics, balance and comfort control are performed so that the resulting motion is not only visually realistic but also dynamically sound. Our approach is to dynamically adjust motions to keep constraint violations at low level.

Our recent work [15] focuses on techniques for animating human locomotion in real-time from simple positional control parameters: the position of the body center and the facing direction. These input parameters are supplied on-line and can be supplied by a human using a VR device or by a program controlling the behavior of a synthetic agent.

Our locomotion methods can switch between five walking modes—walking, running, lateral stepping, backward stepping, or turning around. The appropriate mode is chosen on-line, based on recent input history. When a mode change is necessary, the system generates transitional motion that smoothly blends between the stepping styles.

5. CONCLUSIONS

We believe that physical system simulation and behavior and scenario control are critical supporting technologies for virtual environments. Simulation is a relatively mature discipline, but incorporating simulation capabilities into virtual environment applications presents a number of new requirements and technical challenges. Simulation software must function as a server within virtual environment applications, it must be designed for interaction and run in real-time, and it must support collisions and contacts as natural common phenomena. Furthermore, as virtual environments become richer and more complex, we expect the demand for tools for modeling and authoring scenarios and for controlling the action within VE's to grow dramatically. The work we have described in this paper represents our effort toward meeting the emerging challenges.

Acknowledgements—Paul Henning was the principal developer of the first Isaac implementation. Pete Willemsen and Omar Ahmad developed the current HCSM software. The image of Fig. 4 was created using the Jack software from the University of Pennsylvania. Research by JC was supported in part by Office of Naval Research contract No. N00014-94-1-0576 and NHTSA Cooperative Agreement No. DTNH22-93-YU-07237. Research by JK was supported in part by NHTSA Cooperative Agreement No. DTNH22-93-YU-07237.

REFERENCES


