Timewarp Rigid Body Simulation

Brian Mirtich*

MERL - A Mitsubishi Electric Research Lab

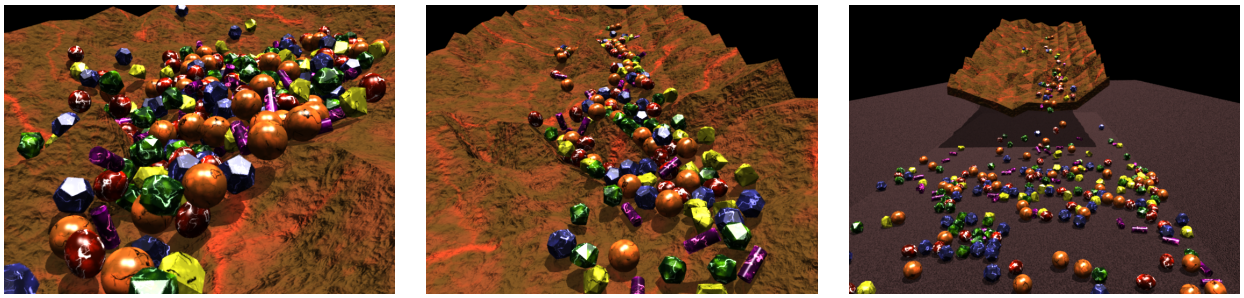

Figure 1: *Avalanche:* 300 rocks tumble down a mountainside.

Abstract

The traditional high-level algorithms for rigid body simulation work well for moderate numbers of bodies but scale poorly to systems of hundreds or more moving, interacting bodies. The problem is unnecessary synchronization implicit in these methods. Jefferson's *timewarp* algorithm [22] is a technique for alleviating this problem in parallel discrete event simulation. Rigid body dynamics, though a continuous process, exhibits many aspects of a discrete one. With modification, the timewarp algorithm can be used in a uniprocessor rigid body simulator to give substantial performance improvements for simulations with large numbers of bodies. This paper describes the limitations of the traditional high-level simulation algorithms, introduces Jefferson's algorithm, and extends and optimizes it for the rigid body case. It addresses issues particular to rigid body simulation, such as collision detection and contact group management, and describes how to incorporate these into the timewarp framework. Quantitative experimental results indicate that the timewarp algorithm offers significant performance improvements over traditional high-level rigid body simulation algorithms, when applied to systems with hundreds of bodies. It also helps pave the way to parallel implementations, as the paper discusses.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation—Continuous, Discrete Event, Animation

Keywords: Physics Based Modeling, Animation.

1 Introduction

Today rigid body simulation is a mature technology. The major components have been well studied and made practical: fast, robust collision detection algorithms [10, 17, 21, 27]; impact models of varying accuracy [8, 12, 31]; methods to enforce general motion constraints [6, 37], especially the ubiquitous non-penetration constraints [3, 4, 35, 36]; and control strategies for articulated bodies [19, 20, 28, 32]. Thus rigid body simulation is available in many animation and CAD packages and used in computer games. Yet areas for significant improvement remain. An important one is increasing the number of moving, interacting bodies that can be simulated.

We are concerned with *general* rigid body simulation, meaning that the bodies have nontrivial geometries, all pairs can potentially collide, and second-order physics governs the motion. There are numerous techniques to simulate large numbers of rigid bodies by relaxing some of these assumptions. Milenkovic efficiently simulates vast numbers of interacting spheres and non-rotating polyhedra using linear programming techniques and zeroeth-order physics [25]. Carlson and Hodgins use different motion levels of detail, from fully dynamic to fully kinematic, to obtain an order of magnitude increase in the number of legged creatures that can be simulated in real time [11]. Chenney *et. al.* cull dynamics computations for off-screen objects; when they enter the field of view initial states are computed by sampling a probability distribution over their state space [13]. Brogan *et. al.* simulate large herds of fully dynamic agents in distributed virtual environments, but without full collision detection [9]. Despite these excellent techniques, the general case is worth pursuing because of its wide applicability; sometimes full collision detection and dynamics cannot be avoided.

Traditional techniques for the general problem become inefficient and even intractable with many-bodied systems for one of two reasons. Either the integration steps[1] become very small, or the amount of work that is wasted because of unpredictable events (like collisions) becomes very large. The problems are not in the component algorithms but in the glue holding them together—the high-level simulation loop. It imposes a synchronization between bodies that is usually unnecessary and wasteful. These problems are explored in depth in Section 2. Jefferson's *timewarp* algorithm [22], discussed in Section 3, is an elegant paradigm designed to alleviate similar problems in parallel discrete event simulation by running processes as asynchronously as possible. An optimistic, non-interaction assumption prevails, and when it is violated only the computation that is provably invalid is undone. Although rigid

* mirtich@merl.com

[1] Throughout this paper, *integration step* means the time interval passed to the integrator, not the smaller steps it may take internally.

body dynamics is a continuous process, it exhibits many traits of a discrete process. With some modification, the timewarp algorithm can be used in rigid body simulators, improving both their speed and scalability. The method is described in Section 4, and Section 5 presents results from an actual implementation.

Timewarp rigid body simulation also supports the long-range goal of a highly parallel implementation. Rigid body simulation offers unlimited potential for modeling the complex and unanticipated interactions of rich virtual environments, but current technology cannot support this. Meeting this challenge will certainly require a multiprocessor approach, with perhaps hundreds of processors computing motion throughout the environment. Such a *simulation farm* is akin to the rendering farms that generate today's high quality computer animation. Section 6 touches on these issues.

2 Simulation Discontinuities

The dominating computation in a rigid body simulator is that of numerically integrating the dynamic states of bodies forward in time. The differential equations of motion have been known for centuries; the true difficulty lies in processing simulation *discontinuities*, here defined as events that change the dynamic states or the equations of motion of some subset of the bodies. Examples include collisions, new contacts, transitions between rolling and sliding, and control law changes. Integrators cannot blithely pass through discontinuities. Instead the integration must be stopped, the states or equations of motion updated, and then the integrator restarted from that point. Compounding this complication is the fact that the times of most discontinuities are impossible to predict. Thus the integration must be interrupted even more frequently than the rate at which discontinuities occur, just to *check* if they have occurred. There are two common approaches for coping with discontinuities, both of which have been shown practical for moderate numbers of bodies.

2.1 Retroactive Detection

Retroactive detection (**RD**) is the most common approach to handling discontinuities. The simulator takes small steps forward and checks for discontinuities after each step [2, 23]. For example, inter-body penetration indicates that a collision occurred at some time during the most recent integration step. A root finding method localizes the exact moment of the discontinuity. After resolution, the integration is restarted from that point. All of the bodies must be backed up to their states at the time of the discontinuity because (1) the discontinuity may have affected their motion, and (2) the bodies directly involved in the discontinuity must certainly be backed up to this time, and there is no framework for maintaining bodies at different times—the bodies must be kept synchronized. The first problem is avoidable by bounding a discontinuity's influence. A certain collision may provably have no influence on the motion of a distant body over the current integration step. However, the second problem is fundamental to **RD**. It does not suffice to maintain states at two different times, the time of the discontinuity and the time at the end of the step, because multiple discontinuities can occur at different times in a single step. Also, earlier discontinuities may cause or prevent later ones, and it is hard to determine which one occurred first without localizing the times of each. In practice, all bodies are backed up to the point of each discontinuity. This method is correct since it eventually processes all real discontinuities and no spurious ones, and Baraff has shown it to be efficient and eminently practical for moderate numbers of interacting bodies [5]. As the number of bodies increases, so does the the rate of discontinuities, and the wasted work per discontinuity increases since more bodies must be backed up. Shrinking the step size to reduce the amount of backup is not a good solution as we shall see. Eventually **RD** becomes intractable due to the amount of wasted work.

2.2 Conservative Advancement

Conservative advancement (**CA**) is an alternative to **RD** based on the idea of never integrating over a discontinuity. Conservative lower bounds on the times of discontinuities are maintained in a priority queue sorted by time, and the simulator repeatedly advances all simulated bodies to the bound at the front of the queue. The simulator tends to creep up to each discontinuity, taking smaller steps as it gets closer. Von Herzen *et. al.* use this approach to detect collisions between time-dependent parametric surfaces [18], and Mirtich uses it to support impulse-based simulation [26]. Snyder *et. al.* use a related approach to locate multi-point collisions by using interval inclusions to bound surfaces in time and space [33]. Finally, **CA** forms the basis for kinetic data structures pioneered by Basche *et. al.* [7]. These are used to solve a host of problems from dynamic computational geometry, such as maintaining the convex hull of a moving point set, by maintaining bounds on when the combinatorial structure may change. For rigid body simulation the advantage of **CA** is that it does not waste work by integrating bodies beyond a discontinuity. Unfortunately, as the number of bodies increases the average time to the next discontinuity check decreases, and the problem is exacerbated since it is difficult to compute tight bounds on times of collisions and contact changes. Stopping the integration of all bodies at each check is very inefficient, and **CA** becomes intractable with many bodies.

2.3 Step Sizes and Efficiency

Figure 2 graphically demonstrates the problem with small integration steps. It shows the computational cost of computing the 10-second trajectory of a ballistic, tumbling brick using a fifth order adaptive Runge-Kutta integrator [30] under various step sizes. The two qualitatively similar curves correspond to different integrator error tolerances. At small step sizes the integrator does not need to subdivide the integration step into smaller pieces to meet the error tolerance. Thus computation is proportional to the number of invocations: halving the step size doubles the work. At large step sizes the integrator breaks the requested step into smaller pieces to meet the error tolerance, so computation is insensitive to step size. Unfortunately, even with a moderate number of bodies, a simulator's operating point is to the left of the elbow in these curves. Thus, reducing the step size significantly increases computational cost.

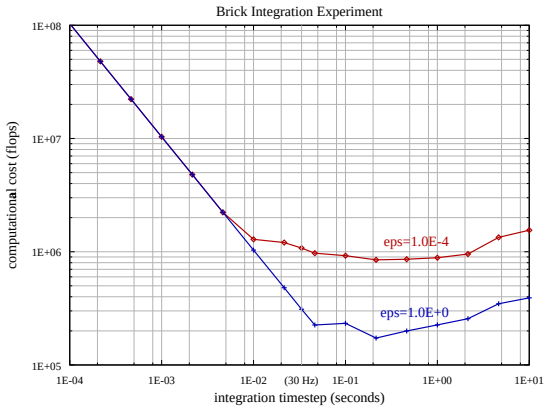

Figure 2: Cost of computing the trajectory of a brick versus integration step size (*eps* is the integrator error tolerance).

3 The Timewarp Algorithm

The problems of **RD** and **CA** result from unnecessary synchronization. Each discontinuity affects only a small fraction of the bodies,

yet under **RD** every body must be backed up when a discontinuity occurs, and under **CA** integration of every body must stop for a discontinuity check. The inefficiencies are tolerable as long as there are not too many bodies. Similar issues arise in discrete event simulation (DES) , which is often applied to very large models such as cars on a freeway system. These simulations are often done in parallel or distributed settings. The simulated agents are partitioned among a number of processors, each of which advances its agents forward in time. There are causality relationships that must be preserved (e.g. a car suddenly braking causes the car behind it to brake), and the crux of the problem is that one agent may trigger an action of another agent on a different processor. Obviously communication by message passing or other means is needed.

Conservative DES protocols guarantee correctness by requiring that each processor advance its agents forward to a certain time only when it has provably received all relevant events from other processors occurring before that time. Optimistic protocols were a key breakthrough in distributed DES. These allow each processor to advance its agents forward in time by *assuming* all relevant events have been received, thereby avoiding idle time. The catch is that when an agent receives an event in its "past," the agent needs to be returned to the state it was in when the event occurred, its own actions since that time must be undone, and the intervening computation is wasted. Jefferson was among the first to define a provably correct, optimistic synchronization protocol along with a simple, elegant implementation called the *timewarp* mechanism [22]. We now give a brief, simplified description of this seminal algorithm.

Each process maintains the state of some portion of the modeled system. Each process also has a local clock measuring *local virtual time (LVT)* at that process. The local clocks are not synchronized, and processes communicate only by sending messages. Every message is time stamped[2] with a time not earlier than the sender's *LVT* but possibly earlier than the receiver's *LVT* when the message is received. Processes must process events in time order to maintain causality constraints. When a received message has a timestamp later than the receiver's *LVT*, it is inserted into an input queue sorted by timestamp. A process's basic execution loop is to advance *LVT* to the time of the first event in its input queue, remove the event, and process it. Advancing to a new time means creating a new state, and these are queued in time order in a state queue.

If the first event in a process's input queue has a receive time earlier than *LVT*, the process performs a *rollback* by returning to the latest state in its state queue before the exceptional event's time. This becomes the new current state, its time becomes the new *LVT*, and all subsequent states in the queue are deleted. Already processed events occurring after the new *LVT* are placed back in the input queue. Messages the processor sent to other processes at times after the new *LVT* are "unsent" via *antimessages*. When a process sends a message, it adds a corresponding antimessage to its output queue. This is a negative copy of the sent message, identical to it except for a flipped sign bit. When a process is rolled back to a new *LVT*, all antimessages in the output queue later than this time are sent. When a message and antimessage are united in a process's input queue, they annihilate one another, and the net effect is as if a message were never sent. Rollback is recursive: antimessages may trigger rollbacks that generate new antimessages.

Global virtual time (GVT) is the minimum of all *LVTs* among the processes and all times of unprocessed messages. It represents a line of commitment during the simulation: states earlier than *GVT* are provably valid while states beyond *GVT* are subject to rollback. Individual *LVTs* occasionally jump backwards, but *GVT* monotonically increases. Since rollback never goes to a point before *GVT*, each state queue needs only to maintain one state before *GVT*. Earlier states as well as saved messages prior to *GVT* may be deleted.

[2]Each message actually has two timestamps, a send and receive time, but one suffices for our purposes.

4 Timewarp Rigid Body Simulation

Rigid body simulation computes a continuous process but exhibits traits of DES. Bodies "communicate" through collisions and persistent contact. Collisions are in fact usually modeled as discrete events. Contact is a continuous phenomenon, but it can be viewed as occurring *within* a collection of bodies rather than between individual bodies. This view facilitates the adaptation of the timewarp algorithm to uniprocessor rigid body simulation. The result is a high-level simulation algorithm that does not suffer from the wasted work problem of **RD** nor the small timestep problem of **CA**.

4.1 Overview

First consider a simulation without connected or contacting bodies. Each body is a separate timewarp process with a state queue containing the dynamic state (position and velocity) of the body at the end of each integration step. The times of these states are different for different bodies. A global event queue contains events for all simulated bodies; this corresponds to a union of all the individual input queues in Jefferson's algorithm. Each event has a timestamp and a list of the bodies that receive it. One iteration of the main simulation loop consists of removing the event from the front of the event queue, integrating the receiving body or bodies to the event time, and then processing the event. Most events are rescheduled after they are processed. Our system supports four types of events:

1. *Collision check events* are received by pairs of bodies, causing a collision check to be performed between them at the given time. Processing these events may lead to collision resolution.

2. *Group check events* trigger collision checking between contacting bodies and also checking for when groups of such bodies should be split. They can also lead to collision resolution.

3. *Redraw events* exist for every rendered body. Processing one involves writing the current position of the body to a recording buffer. Rescheduling occurs at fixed frame intervals.

4. *Callback events* are received by arbitrary sets of bodies and invoke user functions written in *Scheme* that, for example, drive control systems. Rescheduling is user-specified.

4.2 Collisions and Rollback

If penetration is discovered in processing a collision check or group check event, then a collision has occurred at a time preceding the time of the event. This may be a normal collision or a soft collision producing a new persistent contact. Either way, the colliding bodies must be rolled back to the collision time. This behavior differs from that of standard timewarp events which only cause rollback up to the time of the event; it occurs in rigid body simulation because exact collision times cannot be predicted. To implement collision rollback each collision check and group check event has an additional timestamp, a *safe time*, which is the time when the pair or group of bodies was last verified to be disjoint. When a collision check or group check event is processed, and there is no penetration, the safe time is updated to the time of the check. When penetration is detected, the safe time forms a lower bound on the search for the collision time. Since rollback never proceeds to a point before the safe time, *GVT* can be computed as the minimum of all *LVTs* and all event safe times. This insures there are always states to back up to when a collision occurs.

The antimessage mechanism is more general than what is needed for uniprocessor rigid body simulation. Still considering only isolated bodies, the only inter-body communication is through collisions; a suitable record of these drives the rollback. Pairs of cor-

responding post-collision states are linked together, turning the individual state queues into a dynamic *state graph* as shown at the top of Figure 3. The figure depicts the actions taken when bodies A and B collide. Body A is rolled back by deleting all of its states after the post-collision state. (If B also had such states, a twin rollback operation would begin in its own state queue). Some of the deleted states are linked via collisions to states in other bodies. These inter-body communications are now suspect due to the A-B collision, thus rollback proceeds across the collision links and then recursively forward through other bodies' state queues. Upon completion of rollback, all states that were possibly affected by the A-B collision—and no others—are deleted. In this example the rollback invalidates a substantial amount of work. It is an unusual case but one the simulator must be prepared for.

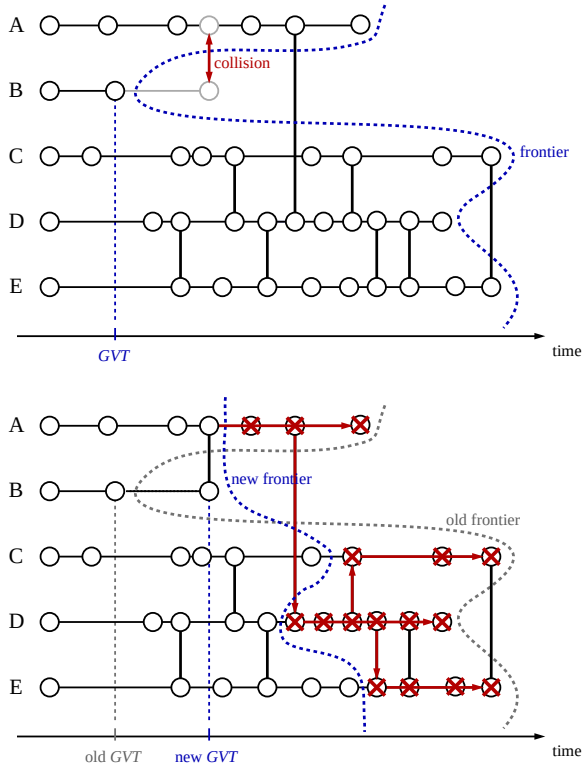

Figure 3: *Top:* State graph of a five body simulation. The vertical connections link post-collision states. The gray states are new post-collision states found while processing an A-B collision check event. *Bottom:* The rollback operation triggered by the collision. Crossed states are deleted and represent wasted work, but forward progress is indicated by the advancement of *GVT*.

Events must also be rolled back. This corresponds to placing messages back in a process's input queue in Jefferson's original algorithm. An event needs to be rolled back only if it involves a body whose state queue was rolled back to a time earlier than the scheduled time of the event. Event rollback is type-specific. Redraw events are simply rescheduled to the first frame time following the rollback time. Fixed-rate callback events are handled similarly. If the rollback time is earlier than the safe time of a collision check or group check event, the event is rescheduled to the rollback time. If the rollback time is between the safe time and the scheduled event time, the system optimistically assumes no action is necessary. This is a gamble since a collision may make the previously computed collision check time inaccurate, but the timewarp algorithm can recover gracefully from poorly predicted collision times.

In total the timewarp algorithm requires little overhead and few additional data structures when compared to a conventional simulator. Any simulator computes sequences of body states; the main change is that these are kept in queues and linked together at the collision points. Rollback is implemented with a simple recursive traversal of the state graph.

4.3 Multibodies

Multibodies (or articulated bodies) are collections of rigid bodies connected by joints, as in a human figure. The trajectory of a single multibody link cannot be determined in isolation; the motion of all links must be computed together. Little change is needed to incorporate multibodies into the timewarp framework. A single state queue serves for the entire multibody; it is advanced as a unit. Most events are still handled on a per rigid body (per link) basis. When, for example, a particular multibody link must be integrated to a certain time for a collision check, the whole multibody is integrated to that time. As a result, states are more densely distributed along multibody state queues than along rigid body state queues, especially for multibodies with many links. A collision involving a single link causes the whole multibody to be rolled back. Clearly timewarp does not offer much improvement if all of the bodies are connected into only a few multibodies.

4.4 Contact Groups

Contact groups are collections of rigid bodies and multibodies in persistent contact; the component bodies exert continuous forces on each other. The components must again be integrated as a unit, but unlike multibodies contact groups are fluid: bodies may join or leave groups, and groups are created and destroyed during a simulation. Contact groups have no analog in the classical timewarp algorithm, which is designed for a static set of processes. Most of the added work in implementing timewarp rigid body simulation is in managing contact groups. To impart some order we require that groups comprise a fixed set of bodies; when the set must change a new group is created. Groups are created by fusions and fissions. A fusion is a suitably soft collision between two bodies, after which they are considered to remain in contact. Either body may be part of a multibody or another group. A fission is a splitting of a group into two or more isolated bodies or separate (non-contacting) groups.

The complexities of contact group evolution are best explained by example. The top of Figure 4 shows the state graph for five rigid bodies labeled A-E and the various contact groups that exist over the time interval $[t_0, t_2]$ (body F does not have a state queue since it is fixed). The bottom of the figure depicts the physical configuration at three distinct times. At time t_0, only bodies B and E are isolated; the others are members of two contact groups, AF and CDF. Only kinematically controlled bodies, of which fixed bodies are a special case, may be members of multiple groups at a given time; such bodies do not link groups together since their motion and the forces they exert on other bodies are independent of the forces exerted on them. Dotted horizontal lines indicate intervals without isolated states since the body is part of a group. The first change after t_0 is a fusion collision between A and B, creating a new group, ABF. D and E then collide, but this is a standard (non-fusion) collision so E remains isolated and CDF intact. The D-E collision does set D in motion, eventually leading to an A-D fusion collision. This latter collision causes two previously separate groups to fuse into a single one, which is the situation at time t_1. Next D breaks contact with C, triggering the fission of $ABCDF$ into $ABDF$ and CF. No collision occurred here; fissions can be caused simply by breaking contacts. Still sliding, D pushes B off of A, causing B to leave the contact group and return to an isolated state. Finally, E lands and settles onto D, fusing into a new group $ADEF$.

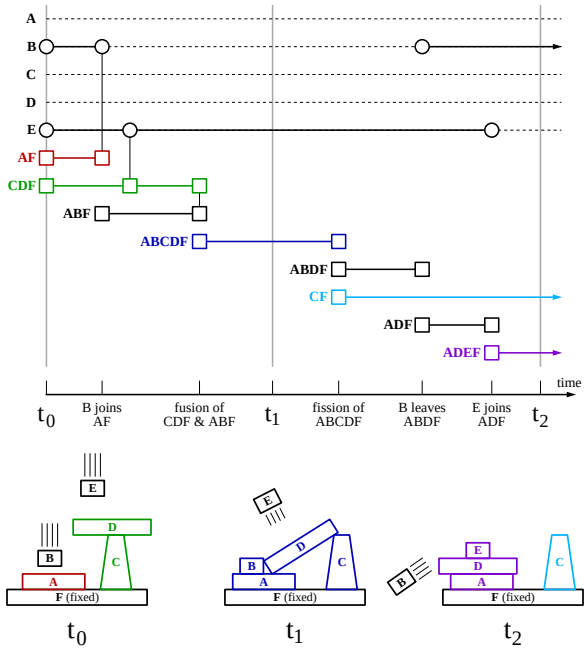

Figure 4: *Top:* The state graph for a portion of a six body simulation. Circles are isolated body states and squares are contact group states. *Bottom:* The physical configuration of the bodies at three distinct times. Moving bodies in contact groups are colored to match the top part of the figure. See text for details.

The state graph in the figure only shows states relevant to the discussion. There would actually be many more states along all of the state queues generated by other events and discontinuities. For example there are usually many non-fusion collisions leading up to a fusion collision as bodies settle. At any time coordinate each non-kinematic body is isolated or a member of exactly one group. Thus there is never ambiguity about what the state of a body is at a given time, or from which state to integrate when computing a new state of a body. To compute the state of body B at time t_2, integration proceeds from the latest isolated state of B prior to t_2. To compute the state of B at time t_1, integration proceeds from the latest state of group $ABCDF$ prior to t_1. To facilitate this, the state graph has additional pointers not shown in the figure. A fusion collision points to the new group it creates, if any. Also, the last state of every fissured group points to the newly isolated bodies and subgroups that succeed it. These pointers make it possible to find for any body B and time t the latest state of B, possibly in a group, prior to t. The search begins within B's own (isolated) state queue and extends into contact groups if necessary by following pointers. Sometimes several pointers and contact groups must be traversed to find the proper prior state. The pointers also facilitate rollback. When a fusion collision state is deleted, the rollback proceeds to the new group formed by the collision, if any. When the last state of a fissured group is deleted, rollback proceeds to the isolated body and subgroup states that succeeded it.

Over the interval shown in Figure 4, six new contact groups are created in addition to the two that existed at t_0. At t_2 only two remain. Groups are terminated when they fuse into new groups or when they fissure into pieces. Termination does not mean the group can be deleted since rollback can cause event processing in non-temporal order. For example it may be necessary to determine the state of body B at time t_1 *after* the group $ABCDF$ is terminated. Once *GVT* passes the last state in a terminated group, however, the group is obsolete and the storage can be reclaimed. A group is also deleted when a rollback operation annihilates all of its states.

Intra-group collision detection is handled in one of two ways. If bodies A and B are in the same group but not currently in contact, the standard A-B collision check event triggers collision detection between them. Each group has a group check event that performs all of the collision detection between already contacting bodies. The distinction is needed since most collision time predictors do not compute meaningful results when the separation distance is near zero. Instead, group check events are scheduled at a fixed, user-specified rate. While collision detection between A and B is being handled by a group check event, the ordinary A-B collision detection event is disabled.

Group check events are also responsible for detecting fissions. A graph is constructed in which the group's non-kinematic bodies are vertices and contacts are edges. A standard connected component algorithm is performed on this graph. Multiple components indicate that the group can be split. There is flexibility in the time to split a group. Integrating a group with multiple connected components does not give a wrong answer; it is simply inefficient since smaller groups can be integrated faster than a single combined one.

4.5 Collision Checks

At any given point in a simulation, collision checking is enabled between certain *active pairs* of bodies, which are hopefully small in number compared to the total number of pairs [21]. Every non-contacting active pair requires a collision check event. The bodies' state queues provide a simple way to keep the number of active pairs small. An axis-aligned bounding box is maintained around the set of states currently computed for each rigid body (hence there are multiple boxes for multibodies and groups). This swept volume grows as new states are computed; it shrinks when states are deleted as *GVT* moves past them. Using six heaps to maintain the minimum and maximum x, y, and z coordinates of the rigid body at each state, the swept volume over n states is updated in $O(\log n)$ time.

The pairs of swept volumes that overlap can be maintained using a hierarchical hash table [29] or by sorting coordinates along the three coordinate axes [3, 14]. If the swept volumes of bodies A and B do not overlap, then A and B are known to be collision free over the interval $[GVT, t_{\min}(A, B)]$, where $t_{\min}(A, B)$ is the time of A's or B's latest state, whichever is earlier. As long as the swept volumes remain disjoint, A and B are not an active pair. Now suppose integration of B causes its swept volume to overlap the previously disjoint swept volume of A. To avoid missing collisions, a new collision check event for A and B is scheduled for the time given by the value of $t_{\min}(A, B)$ *before* B was integrated (Figure 5). The bodies are known to be collision free before this point. This new event is in B's past, but the timewarp algorithm can accommodate it; if a collision did occur then rollback will rectify the situation. The collision check event for A and B remains active as long as their swept volumes overlap. This method works even though the swept volumes exist over different time intervals and may have no states at common times. Inactive pairs do not need to be synchronized in order to remain inactive, which avoids costly integration interruptions for the vast majority of body pairs.

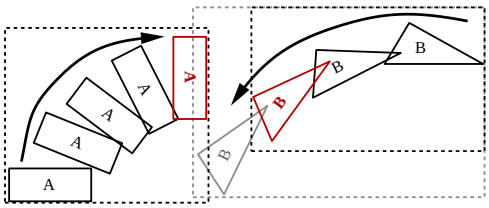

Figure 5: When B is integrated to the state shown in gray, swept volume overlap occurs. A and B become an active pair, and a collision check is scheduled at the earlier time of the two red states.

simulation	simulation duration (s)	# of rigid bodies moving/total	# of discontinuities (thousands)	avg time between discontinuities (ms)	avg integr'n step (ms)	# of integr'ns (millions)	total integr'n / moving body (s)	total rollback / moving body (s)	comp time / frame (s)
atoms	120	302 / 308	51.9	2.31	6.25	6.04	125 (+4.2%)	0.278 (0.23%)	0.767
cars	60	428 / 524	17.8	3.38	14.9	1.98	69.2 (+15%)	1.57 (2.6%)	0.904
robots	120	240 / 430	26.8	4.48	9.88	3.00	124 (+3.3%)	1.45 (1.2%)	0.707
avalanche	45	300 / 824	217	0.208	3.39	5.84	66.0 (+47%)	7.15 (16%)	97.0

Table 1: Data collected over the four simulations.

4.6 Callback Functions

It is difficult to completely hide the underlying timewarp nature of the system from user callback functions. Because the bodies' *LVTs* are not synchronized, callback functions involving different bodies are not invoked in strict temporal order. In fact, a callback for a single body may not be invoked at monotonically increasing times due to rollback. Thus, a collision callback that counts a body's collisions by incrementing a global counter is flawed since it may get called with the same collision multiple times. One convention that guarantees correct behavior is to forbid callback functions from accessing global data. The function should only use the data passed in: the time of the event and the states of the relevant bodies at that time. Data that must persist across callback invocations are supported by adjoining new slots to the states of bodies. Unlike position and velocity values, the values in these slots are simply copied from state to state since there is no need to integrate them, but callback functions can access and modify these values. Changes are appropriately undone when the state queues are rolled back. The collision counter is implemented correctly by attaching an integer slot to the body state. The callback function increments the counter, and rollback may cause the counter to decrease.

5 Results

We now describe the results of simulating four different systems with a timewarp rigid body simulator (Figures 1 and 7). Our implementation draws from a myriad of component algorithms and techniques described in the literature; Appendix A describes the major ones. Robustness—always an issue in rigid body simulation—is paramount for the kinds of simulations studied here. Anything that can go wrong certainly will when simulating large systems over long times. Our implementation favors robustness over efficiency. The issues are not the underlying components nor the absolute efficiency of this particular implementation but the degree to which timewarp improves any implementation's performance.

Atoms simulates 200 spheres and 100 water-like molecules bouncing in a divided box. During the simulation the divider compresses one compartment and lifts to allow the gasses to mix. *Cars* simulates four multibody vehicles with active wheel velocity and steering angle controllers. These drive over a course with speed bumps and an array of 400 spherical pendulums. *Robots* simulates 20 eight-link manipulators that repeatedly pick up boxes and throw them. The robots are fully dynamic objects, controlled via joint torques commanded by callback functions. Callbacks also use an inverse kinematic model for motion planning. Finally *avalanche* simulates 300 rigid bodies tumbling down a mountainside, creating a vast number of interactions. With the exception of *atoms*, all simulations use realistic values for length, mass, time and earth gravity. Each was generated from a single run.

5.1 Full Timewarp Simulation Data

Table 1 shows data collected over the course of performing the full simulations. The percentages in the total integration and rollback columns are with respect to the simulation duration. Computation times were measured on an SGI Onyx (200MHz R10000 CPU). Integration and rollback intervals of multibodies and groups were weighted by the number of individual rigid bodies involved. The reason that total integration minus rollback exceeds duration is because of the added integration involved in localizing discontinuities. When a discontinuity is detected over an interval, the simulator must compute new states of the relevant bodies in order to localize it. This means re-integrating over certain time intervals, increasing the total integration time.[3]

Worth noting is the amount by which the average integration step exceeds the average interval between discontinuities. This of course is a key advantage of the timewarp algorithm: integration of a body does not halt at every discontinuity but only at the ones which are relevant to it. The fact that the actual integration steps are 2–16 times larger than the average interval between discontinuities is especially noteworthy since any simulation strategy (**RD**, **CA**, or timewarp) must *check* for discontinuities at a much higher rate than they actually occur. In our experiments, checks outnumbered actual discontinuities by two orders of magnitude. Under **RD** or **CA**, all bodies are halted at every check, although the problem is less severe under **RD** since collision checks are synchronized. Table 1 also shows that rollback is a modest cost. Through judicious undoing, timewarp avoids the large amount of wasted work inherent in **RD** as the number of bodies increases.

In several performance measures, the *avalanche* simulation is an outlier. The slow simulation speed is not because timewarp is not working. The ratio of average integration step to average time between discontinuities is quite good, and the total integration per body, while high, is not prohibitive. The main difficulty is the complexity of the contact groups: over 16,000 groups are formed, some having as many as 64 moving bodies and 217 simultaneous contacts. Simulating an avalanche using particle or position-based physics may be more practical, but the example shows that timewarp can handle even extreme cases well.

5.2 Comparative Simulation Data

Table 1 suggests the timewarp algorithm is a good idea. Further experiments give a more quantitative measure of the improvement it brings. We added alternate main loops to the simulator to let it use **RD** and **CA** policies instead of timewarp (**TW**). The **RD** algorithm is parameterized by the basic timestep to attempt on each iteration; we used values of 0.001, 0.01, and 1/30 second. All five algorithms were run on a two-second segment of an *atoms* simulation, with the divider stationary in the middle of the box and with the number of bodies varying from 25 to 200. The upper part of Figure 6 shows the average integration step taken by the simulator under the various algorithms. The results confirm the key problem with **CA**: as the number of bodies increases the average time to the next discontinuity check decreases. As Figure 2 shows, the small steps

[3] Baraff cleverly avoids this waste by using internal values of the Runge-Kutta integrator to obtain a polynomial approximation of the state over an integration step for free [5].

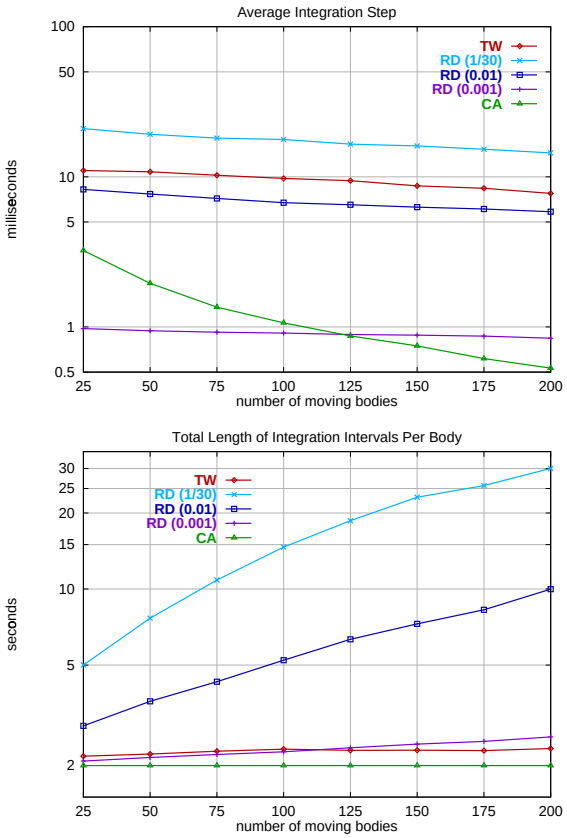

Figure 6: Integration statistics for various *atoms* simulations.

have a drastic effect on computational cost. **RD**'s average timestep is not as sensitive to the number of bodies since it always tries to take a fixed size step forward. **TW** is also not sensitive to it since the bodies are decoupled. The lower portion of the figure exposes the problem with **RD**: wasted work. For a two-second, 200-body simulation and a frame rate timestep, **RD** integrates each body an average of 30 seconds. This is to be compared with **TW**'s value of 2.3 seconds and the modest percentages in the *total integration* column of Table 1. **CA** never integrates more than two seconds per body since it uses a one-sided approach to each discontinuity.

Actual execution times shed further light. For 100 atoms, **RD-1/30** is the narrow winner at 0.142 s/frame, while **TW** was 0.147 s/frame and **CA** was 1.15 s/frame. By 200 atoms, **TW** is clearly superior at 0.388 s/frame, while **RD-0.01**, the fastest **RD** algorithm, was 2.61 s/frame, and **CA** was 4.74 s/frame.

6 Conclusion

Timewarp rigid body simulation is clearly able to simulate larger systems with more interactions than traditional synchronized simulation algorithms. The most obvious avenues for future research involve parallel rigid body simulation. Timewarp simulation helps pave the way to this goal since the individual bodies are evolved asynchronously. If the algorithm runs on multiple processors, delays due to communication latencies are handled in the same way as bad predictions of discontinuity times: with minimal rollback.

The simplest way to structure a parallel simulator would be to have one master processor that repeatedly sends integration tasks to a bevy of slave processors. All of the global data structures could be kept on the master processor, requiring little change in the algorithms presented here. This could significantly boost performance over the uniprocessor case but suffers from a bottleneck at the master. An egalitarian approach in which bodies are distributed among processors is ultimately more scalable. Important open questions are how to parcel the bodies among processors and how to balance workloads. At odds are the goals of minimizing inter-processor communication by keeping bodies in the same spatial region on a common processor and minimizing idle time by shifting bodies to idle processors. At any rate some method and strategy for migrating the bodies between processors seems appropriate. Rollback probably requires a full antimessage mechanism since the state graph is likely to be distributed. Other questions surround how and where to store data structures like the spatial hash table, which is frequently accessed by all bodies. Events involving multiple bodies might be redundantly stored and processed on multiple processors or on only one of the relevant processors. Finally, there are various protocol choices for passing state information between processors. Clearly there are many challenges to building a large simulation farm. Yet the prospect of rich virtual environments built on a physics-based substrate is adequate motivation to pursue them.

A Implementation Details

Our system is implemented in C++. All geometries are modeled as convex polyhedra or unions thereof. The *v-clip* algorithm [27] is used for narrow-phase collision detection; a hierarchical spatial hash table [26, 29] containing axes-aligned bounding boxes is used for the broad phase. For nearby bodies not in contact, times to impact are estimated from current positions and velocities as in [26], but the predictions are not conservative. Persistent contact is modeled using a penalty force method; spring and damper constants are specified per body pair. Inspired by [1] we use an implicit integrator (4th order Rosenbrock [30]) with a sparse solver [15] to handle stiffness induced by the penalty method. This is only necessary for contact groups; isolated bodies are integrated with a 5th order Runge-Kutta integrator [30]. We use a smooth nonlinear friction law, $f_t/f_n = \tanh(v_t/\varepsilon)$ [34]; static friction is not modeled. Reduced coordinates are used for multibodies, with dynamics computed by a generalized Featherstone algorithm [16] in the isolated case and by the spatial composite-rigid-body algorithm [24] within contact groups. The latter is more suited to generating the acceleration Jacobian required by the implicit integrator.

References

[1] D. Baraff and A. Witkin. Large Steps in Cloth Simulation. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, July 1998.

[2] David Baraff. Curved Surfaces and Coherence for Non-Penetrating Rigid Body Simulation. In *Computer Graphics (SIGGRAPH 90 Conference Proceedings)*, volume 24, pages 19–28. August 1990.

[3] David Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. Ph.D. thesis, Department of Computer Science, Cornell University, March 1992.

[4] David Baraff. Fast Contact Force Computation for Nonpenetrating Rigid Bodies. In *SIGGRAPH 94 Conference Proceedings*, Annual Conference Series, pages 23–34. ACM SIGGRAPH, Addison Wesley, 1994.

[5] David Baraff. Interactive Simulation of Solid Rigid Bodies. *IEEE Computer Graphics and Applications*, 15(3):63–75, May 1995.

[6] Ronen Barzel and Alan H. Barr. A Modeling System Based on Dynamic Constraints. In *Computer Graphics (SIGGRAPH 88 Conference Proceedings)*, volume 22, pages 179–188. August 1988.

[7] J. Basch, L.J. Guibas, and J. Hershberger. Data Structures for Mobile Data. In *Proceedings of 8th Symposium on Discrete Algorithms*. 1997. To appear in J. of Algorithms.

[8] Raymond M. Brach. *Mechanical Impact Dynamics; Rigid Body Collisions*. John Wiley & Sons, Inc., 1991.

[9] David C. Brogan, Ronald A. Metoyer, and Jessica K. Hodgins. Dynamically Simulated Characters in Virtual Environments. *IEEE Computer Graphics and Applications*, 18(5):58–69, September 1998.

[10] Stephen Cameron. Enhancing GJK: Computing Minimum Penetration Distances between Convex Polyhedra. In *Proceedings of International Conference on Robotics and Automation*. IEEE, April 1997.

[11] Deborah A. Carlson and Jessica K. Hodgins. Simulation Levels of Detail for Real-time Animation. In *Proc. of Graphics Interface '97*, pages 1–8. 1997.

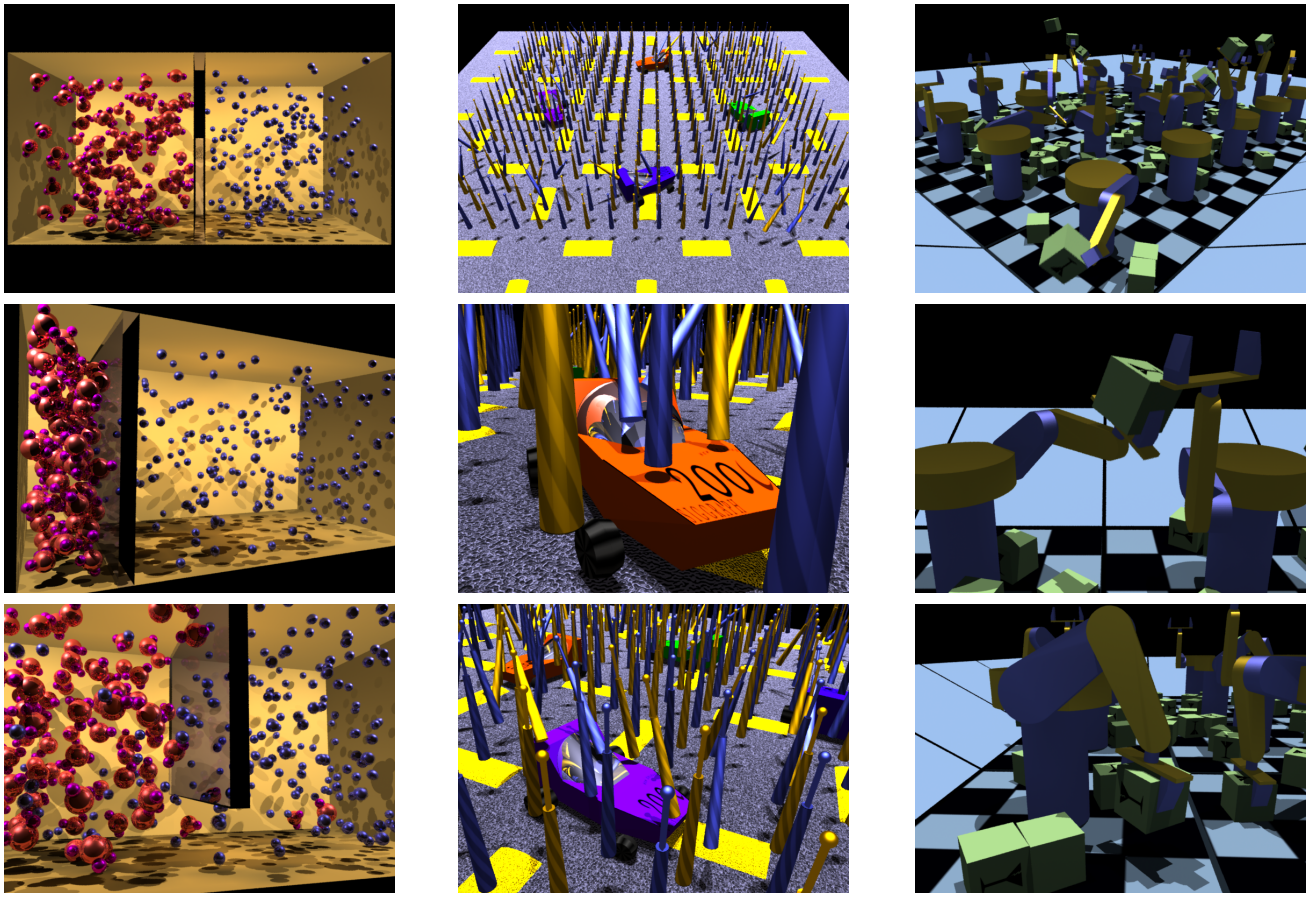

Figure 7: *Left* to *Right*: snapshots from the *atoms*, *cars* and *robots* simulations (thanks to Larry Gritz for *Blue Moon Rendering Tools*).

[12] Anindya Chatterjee and Andy Ruina. A New Algebraic Rigid Body Collision Law Based on Impulse Space Considerations. *Journal of Applied Mechanics*, 65:939–951, December 1998.

[13] Stephen Chenney, Jeffrey Ichnowski, and David Forsyth. Dynamics Modeling and Culling. *IEEE Computer Graphics and Applications*, 19(2):79–87, March/April 1999.

[14] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav K. Ponamgi. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments. In *Symposium on Interactive 3D Graphics*, pages 189–196. ACM SIGGRAPH, April 1995.

[15] J. Dongarra, A. Lumsdaine, R. Pozo, and K. Remington. A Sparse Matrix Library in C++ for High Performance Architectures. In *Proceedings of the Second Object Oriented Numerics Conference*, pages 214–218. 1992. `www.math.nist.gov/iml++`.

[16] R. Featherstone. The Calculation of Robot Dynamics Using Articulated-Body Inertias. *International Journal of Robotics Research*, 2(1):13–30, 1983.

[17] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, August 1996.

[18] Brian Von Herzen, Alan H. Barr, and Harold R. Zatz. Geometric Collisions for Time-Dependent Parametric Surfaces. In *Computer Graphics (SIGGRAPH 90 Conference Proceedings)*, pages 39–48. 1990.

[19] Jessica K. Hodgins and Nancy S. Pollard. Adapting Simulated Behaviors for New Characters. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 153–162. ACM SIGGRAPH, Addison Wesley, August 1997.

[20] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating Human Athletics. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 71–78. ACM SIGGRAPH, Addison Wesley, 1956.

[21] Philip M. Hubbard. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. *ACM Transactions on Graphics*, 15(3), July 1996.

[22] David R. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[23] V. V. Kamat. A Survey of Techniques for simulation of Dynamic Dynamic Collision Detection and Response. *Computer Graphics in India*, 17(4):379–385, 1993.

[24] Kathryn W. Lilly. *Efficient Dynamic Simulation of Robotic Mechanisms*. Kluwer Academic Publishers, Norwell, 1993.

[25] Victor J. Milenkovic. Position-Based Physics: Simulating the Motion of Many Highly Interacting Spheres and Polyhedra. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 129–136. ACM SIGGRAPH, Addison Wesley, August 1996.

[26] Brian Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. Ph.D. thesis, University of California, Berkeley, December 1996.

[27] Brian Mirtich. *V-Clip:* Fast and Robust Polyhedral Collision Detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998. Mitsubishi Electric Research Lab Technical Report TR97–05.

[28] J. Thomas Ngo and Joe Marks. Spacetime Constraints Revisited. In *SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, pages 343–350. ACM SIGGRAPH, Addison Wesley, 1993.

[29] M. Overmars. Point Location in Fat Subdivisions. *Information Processing Letters*, 44:261–265, 1992.

[30] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian R. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, second edition, 1992.

[31] Edward J. Routh. *Elementary Rigid Dynamics*. Macmillan, London, 1905.

[32] Karl Sims. Evolving Virtual Creatures. In *SIGGRAPH 94 Conference Proceedings*, Annual Conference Series, pages 15–22. ACM SIGGRAPH, 1994.

[33] John M. Snyder, Adam R. Woodbury, Kurt Fleischer, Bena Currin, and Alan H. Barr. Interval Methods for Multi-Point Collisions between Time-Dependent Curved Surfaces. In *SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, pages 321–333. ACM SIGGRAPH, Addison Wesley, 1993.

[34] Peng Song, Peter R. Kraus, Vijay Kumar, and Pierre Dupont. Analysis of Rigid Body Dynamic Models for Simulation of Systems with Frictional Contacts, June 1999. Submitted to ASME Journal of Applied Mechanics.

[35] D.E. Stewart and J.C. Trinkle. An Implicit Time-Stepping Scheme for Rigid Body Dynamics with Inelastic Collisions and Coulomb Friction. *International Journal of Numerical Methods in Engineering*, 39:2673–2691, 1996.

[36] J.C. Trinkle, J.S. Pang, S. Sudarsky, and G. Lo. On Dynamic Multi-Rigid-Body Contact Problems with Coulomb Friction. *Zeitschrift fur Angewandte Mathematik und Mechanik*, 77(4):267–279, 1997.

[37] Andrew Witkin, Michael Gleicher, and William Welch. Interactive Dynamics. *Computer Graphics*, 24(2):11–22, March 1990.