Time Warp Rigid Body Simulation
A Technical Report

Shashank Reddy Chirra
International Institute of Information Technology, Bangalore
IMT2018069
Shashank.Reddy@iiitb.org

***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. This is due to unnecessary synchronization between the rigid bodies which is implicit in the algorithm itself. Jefferson's timewarp algorithm [1] is a technique for alleviating this problem in parallel discrete event simulation. Although rigid body motion is a continuous motion, can be very closely modelled by as discrete. 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. In this report we will first introduce the problem of rigid body simulation, we then look at related models to simulate rigid body motion, after this we will look into the detail of the algorithm. We will then look at the quantitative performance of the algorithm.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Today rigid body simulation is a mature technology. The major components have been well studied and made practical. However, these traditional methods become inefficient and even intractable with many-bodied systems for one of two reasons - either the integration steps (integration step means the time interval passed to the integrator, not the smaller steps it may take internally, for example, a small intergration step can mean computing the future position of all the objects for every milli second in the object's time, however a larger step would compute the position for the next 10 seconds) become very small, or the amount of work that is wasted because of unpredictable events (like collisions) becomes very large.

A. 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.

B. Some already known algorithms to Simulate Rigid Body Motion

We will briefly discuss 2 algorithms that simulate rigid bodies to see where they lack and how the timewarp algorithm is an improvement.

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], [3]. When a discontinuity is detected, a root finding method localizes the exact time of the discontinuity. After this, all bodies must be rolled back to their states at the time of the discontinuity, after this all the required states are updated and the simulation continues. We can clearly see that this has a huge overhead as even objects that are very far away and are not related to the scene must be rolled back, this will reduce the efficiency when the number of bodies in the scene are high. Secondly, this also puts a constraint on the step size as large step size will mean that discontinuities will be discovered at each step.

C. 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. 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.

II. Related Work

There are many fast, robust collision detection algorithms [4], [5], impact models of varying accuracy [6], [7] and methods to enforce general motion constraint [8], [9]. 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 [10]. 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 [12]. Despite these excellent techniques, the general case is worth pursuing because of its wide applicability; sometimes full collision detection and dynamics cannot be avoided.

III. Method

We will first look at the Time Warp Algorithm [1], then we will see how it can be applied in the context of rigid body motion.

A. Time Warp Algorithm

Time Warp algorithm ensures that many processes run concurrently over multiple processors however they are not synchronized. 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 with a time not earlier than the sender's LVT. 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. 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.

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.

Thus, we can already see that if a process were to calculate the motion of an object, we can see that rollbacks are localized and hence save unnecessary computation.

B. Time Warp Rigid Body Simulation

If we say that a process governs the state of a body, then we can say that bodies "communicate" through collisions and persistent contact. This view facilitates the adaptation of the timewarp algorithm to uniprocessor rigid body simulation.

We can now get an overview on how this algorithm is implemented,

1) 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 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. 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. Pairs of corresponding post-collision states are linked together, turning the individual state queues into a dynamic state graph which can be traversed to implement the roll back. In total the time warp 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.

2) 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. 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.

3) 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. To impart some

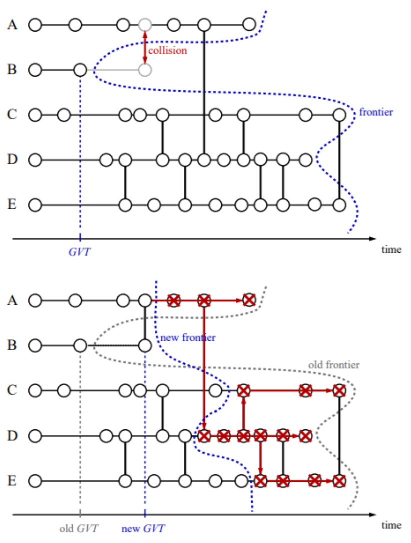

Fig. 1. Rollback implemented through State Graphs.

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.

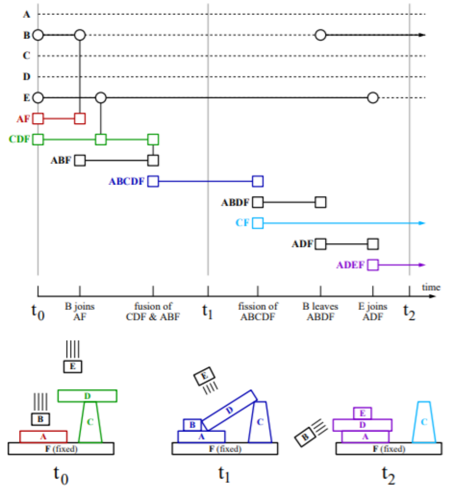

Fig. 2. Top: The state graph for a portion of a six body simulation. Circles are isolated body states and squares are contact group states. Bottom: The physical configuration of the bodies at three distinct times. Moving bodies in contact groups are colored to match the top part of the figure. See text for details.

4) Collision Detection: 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 [13]. Every noncontacting 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 logn time.

As long as the swept volumes remain disjoint, bodies A and B are not an active pair. Now, suppose an integration over B causes their swept volumes to intersect, To avoid missing collisions, a new collision check event for A and B is scheduled for the time given by the value of min(LVT(A), LVT(B)) before the B was integrated. 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.

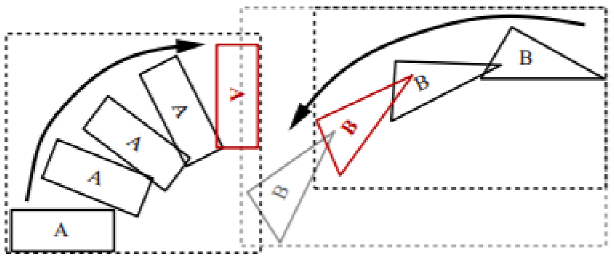

Fig. 3. When B is integrated to the state shown in gray, swept volume overlap occurs. A and B become an active pair, and a collision check is scheduled at the earlier time of the two red states.

5) 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. 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.

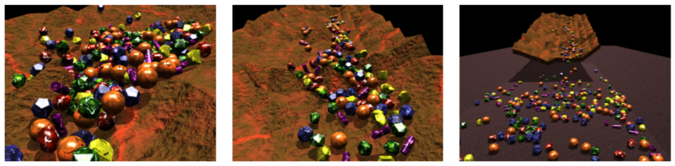

Fig. 4. Avalanche: 300 rocks tumble down a mountainside

IV. Discussion

A. Results of TimeWarp Rigid Body Simulation

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. Finally avalanche simulates 300 rigid bodies tumbling down a mountainside, creating a vast number of interactions.

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 several performance measures, the avalanche simulation is an outlier, 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, however the ratio of the size of the integration step to the average interval between discontinuities was still good, indicating that the time warp algorithm generalizes well even in extreme cases.

V. Conclusion

One obvious avenue for future research is to except this concept to a parallel rigid body simulation, that runs on multiple processors, where closely related bodies are simulated on the same processor. Consequently, this was the subject of research for John Koenig *et al* [14].

The uni-processor Time warp rigid body simulation has since been used in multiple complex simulations such as A Robot Soccer Simulator [15] which focuses on contact problems of rigid body simulation, Modal warping: real-time simulation of large rotational deformation and manipulation [16] which is a real-time simulation technique for large deformations, DyRT: dynamic response textures for real time deformation simulation with graphics hardware [17] which focuses on simulating geometrically complex, interactive, physically-based, volumetric, dynamic deformation models with negligible main CPU costs, and many more.

The time warp rigid body simulation algorithm has also been used in developing new algorithms such as Backward steps in rigid body simulation [19] which focuses on methods to allow time stepping in the backward direction in rigid body simulations, and Adaptive merging for rigid body simulation [18], which focuses on reducing computation time in rigid body simulations by merging collections of bodies when they share a common spatial velocity.

Thus given the frequent use of rigid bodies simulation in movies, video games, and other graphical applications, we can see the need to make this process as efficient as possible and this paper has helped us make a significant leap in this direction.

References

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

[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] 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

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

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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

[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.

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

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

[14] Object-Centric Parallel Rigid Body Simulation With Timewarp John Koenig, Ioannis Karamouzas, Stephen J. Guy Department of Computer Science and Engineering University of Minnesota.

[15] A Robot Soccer Simulator: A Case Study for Rigid Body Contact Eric Larsen Sony Computer Entertainment America RD

[16] Min Gyu Choi; Hyeong-Seok Ko : Modal warping: real-time simulation of large rotational deformation and manipulation

[17] Doug. L.James, Dinesh .K. Pai: DyRT: dynamic response textures for real time deformation simulation with graphics hardware

[18] Eulalie Coevoet, Otman Benchekroun, et al. : Adaptive merging for rigid body simulation

[19] Christopher D. Twigg, Doug. L.James : Backward steps in rigid body simulation.

[20] Brian Mirtich : Timewarp Rigid Body Simulation