

Techniques for Real-Time Rigid Body Simulation

by

Kevin Egan

A Thesis submitted in partial fulfillment of the requirements for Honors
in the Department of Computer Science at Brown University

Providence, Rhode Island

May 2003

© Copyright 2003 by Kevin Egan

This thesis by Kevin Egan is accepted in its present form by
the Department of Computer Science as satisfying the research requirement
for the awardment of Honors.

Date _____

John F. Hughes, Reader

Date _____

Andries van Dam, Reader

Acknowledgements

I owe my readers, John F. Hughes and Andries van Dam, for my introduction, and subsequent education in computer graphics. I also owe Morgan McGuire a great deal for giving me a chance to work with him and gain valuable research experience. Finally, I would like to thank all of my professors, classmates, friends, and family.

Contents

1	Introduction	1
1.1	Preamble	1
1.2	Thesis Organization	2
2	Previous Work	3
2.1	Basic Colliding Technique	3
2.2	Techniques in Video Games	3
2.3	Soft Body Simulation	4
2.4	Miscellaneous High End Techniques	5
2.4.1	Brittle Fracture	5
2.4.2	Probabilistic Sampling	5
2.4.3	Timewarp Integration	6
2.5	Standard Real-Time Techniques	6
2.5.1	Penalty Force Response	6
2.5.2	Impulse Based Response	6
2.6	Analytic Rigid Body Simulation	7
2.6.1	Original Formulation	7
2.6.2	Gauss' Principle of Least Constraints	7
2.6.3	Position Based Time-Stepping	7
2.6.4	Energy Based	8
2.6.5	Articulated bodies	8
2.7	Position Optimizing Techniques	8
2.7.1	Potential Energy Minimization	8
2.7.2	Optimization Based Animation	9

3	Implementation	10
3.1	Simulation Domain	10
3.2	Collision Detection	10
3.3	Collision and Contact Response	11
3.3.1	Analytic Force Determination	11
3.3.2	Impulse Based	12
4	Basic Techniques	14
4.1	Spatial Subdivision	14
4.2	Visual Plausibility	14
4.3	Sticky Collisions	15
4.4	Fixing Interpenetration	15
4.5	Computational Stability	16
5	Novel Techniques	17
5.1	Modified Timewarp	17
5.2	Damped Integration	18
5.3	Damping	19
5.4	Geometric Stability	20
5.5	Filtering	22
6	Results	24
6.1	Damped Integration	24
6.2	Exterior Rotation Problem	24
6.3	Future Work	26
6.3.1	Novel Techniques	26
6.3.2	Rigid-Body Research	26
6.4	Conclusion	27
A	Pseudo Code	30
A.1	Modified Timewarp	30
A.2	Damped Integration	32
	Bibliography	35

Chapter 1

Introduction

1.1 Preamble

In the past decade, the quest for visual realism in interactive 3D computer applications has fueled the large growth of the video game industry and the graphics card industry, as well as many advances in rendering technology. As the visual display does a better job of making users feel immersed within a virtual environment, users will innately expect their environment to not only look realistic, but also to react in a realistic manner. The breakneck pace at which visual realism has increased has highlighted the relatively slow development in other areas, (physical realism, social realism, historical realism, etc.). As an example, if a user is totally immersed in a program and sees a perfect digital rendering of a human, the user will naturally expect to be able to go up and talk to the human. If the user is slightly more anti-social, the user may expect to be able to punch the approaching character and watch the character fall backwards onto the ground.

In particular, this thesis focuses on techniques for incorporating physical realism into interactive 3D computer applications. A simple example is the animation of a box sliding across a table and tumbling over the edge. Surprisingly, this type of interaction is available in almost no commercial software. While anybody that has taken a high school physics course believes they can calculate acceleration and velocity for simple 2D rigid bodies, this seemingly simple task becomes complicated when multiple objects rest on top of each other. Doing this in 3D makes the math more complex. Determining the significant contact points, the forces on those contact points, and the final force and torque applied to the center of mass for a box sliding off the corner of a table is non-trivial.

Many techniques for rigid body simulation have been put forward dating back to 30 years

ago. Real-time applications, however, rarely incorporate general rigid body simulation. This thesis aims to develop new techniques for rigid body simulation as well as provide a fully implemented and viable physics engine.

1.2 Thesis Organization

Chapter 2 presents previous work and explains the current areas, algorithms, and challenges for real-time rigid body simulation. Chapter 3 discusses details of our implementation. Chapter 4 discusses the use of some basic techniques for rigid body simulation. Chapter 5 presents the novel techniques gleaned from our research. Chapter 6 discusses future work and the final results of this research.

Chapter 2

Previous Work

We will explain what we call a basic technique and then describe more robust methods as extensions to this basic technique. We will also explain the motivation for, and drawbacks to, each technique. When discussing interactions we differentiate between high speed collision interactions, which are usually modeled as instantaneous, and resting contact interactions, which are usually modeled as persistent. Some techniques treat both collision and contact interactions equivalently.

2.1 Basic Colliding Technique

The basic technique involves integrating velocity and force forward in time to obtain new positions and velocities. If two objects are found to intersect, the simulation is stopped. The simulator then searches backward and forward in time to find a more exact collision time for the two objects. Once a collision is found, all object positions are rolled back to the time of the collision. The integration proceeds forward at that point. This basic technique only handles collisions and does not handle objects that remain in resting contact.

2.2 Techniques in Video Games

Video games often use very specialized system, but the requirements for these systems are fairly uniform. Physics engines within video games are constrained to be completely consistent both in terms of performance and outcome. A physics engine that can occasionally slow to a crawl or occasionally cause very unbelievable phenomenon is unacceptable for most video games. True rigid body simulation is often replaced by simple logic that executes a fixed animation and then eventually removes “interacting” objects. Examples of this type

of simulation are exploding barrels or knocked cones that quickly disappear or fly off the screen within a racing game.

Some of the more compelling physics engines are worth noting. Recent racing games such as *Gran Turismo 3: A-Spec*¹ include very accurate modeling of force, torque and friction on the individual tires of each car. Action games such as *Halo*² and *Grand Theft Auto: Vice City*³ feature the ability to drive various vehicles off of ramps or other terrain. Previews for the upcoming game *Doom III*⁴ feature the ability to shoot over boxes that tumble into each other. Unfortunately, it is currently impossible to tell exactly what system is being used in *Doom III* and how robust the physics simulation will be. *Trespasser*⁵ may be a large part of why more general techniques are not used in interactive applications. *Trespasser* features a physics system that is designed to accurately model boxes with arbitrary velocity, rotation, size and friction [40]. Unfortunately, the physics engine is slow and has problems with small boxes, and the friction system can add energy to the system causing boxes to vibrate with increasing intensity.

2.3 Soft Body Simulation

Soft body simulation aims to simulate cloth and other semi-flexible materials. Some of the simulation techniques used with soft or deformable bodies are applicable to areas of rigid body simulation. Cloth is often modeled as a collection of rigid hinge constraints that prevent the cloth from tearing but let it fold and bend. Some of the latest cloth simulations deal with how to solve these stiff differential equations and ways of discretizing the models [11].

During the 2001 Game Developer's Conference, Thomas Jakobsen presented a simulation technique [22] for cloth as well as other soft bodies, (in this case dead human bodies for *Hitman: Codename 47*⁶). Jakobsen's technique is radical in its simplicity. In this technique, all physical interactions are modeled with constraints. For instance a rod constraint dictates that two points must remain a fixed distance apart. If one of the points is moved so that this constraint is broken, the constraint is restored in a relaxation step. After iteratively

¹*Gran Turismo 3: A-Spec* copyright Polyphony Digital

²*Halo* copyright Bungie Software

³*Grand Theft Auto: Vice City* copyright Rockstar Games

⁴*Doom III* copyright id Software

⁵*Trespasser* copyright Dreamworks Interactive

⁶*Hitman: Codename 47* copyright Eidos Interactive

correcting the set of broken constraints, Jakobsen reports that most constraints will be met or be “close enough” to appear plausible. This technique is stable, but is not very physically accurate since no care is taken to conserve energy or momentum. With this simulation a ball will not bounce but instead will lose all velocity the moment it collides with a surface. This method seems to be ideal for modeling flexible and absorbing bodies.

2.4 Miscellaneous High End Techniques

2.4.1 Brittle Fracture

Most rigid body simulation makes the assumption that objects are infinitely strong and cannot break. One area where this assumption breaks down is in the simulations of exploding buildings or other destructive events. Being able to simulate and animate these events can be very useful for producing special effects often needed for films. For accurate modeling of fracturing, as well as elastic and plastic deformation, O’Brien and Hodgins model the internal stress and strain of small tetrahedral volumes within objects [31]. Because of the use of penalty methods, many interacting elements, and wildly varying forces this simulation technique is fairly far from interactivity. Simulations can commonly require on the order of 1,000,000 time steps per second to maintain stability.

2.4.2 Probabilistic Sampling

Many simulations often try to find the “correct” answer. In reality, real objects and surfaces have tiny surface details that are usually not modeled in the simulation. Consequently, the “correct” answer according to the simulation may look very artificial and implausible to a human viewer. Barzel et al. note that physical simulations often look more plausible if some randomness is added to the computation [12]. A real ball dropped straight down onto a flat floor will usually not bounce straight back up. This behavior is due to imperceivable spin on the ball and surface variation on the floor.

Chenney and Forsyth add randomness to the simulation to try to achieve an animation that satisfies a list of constraints [13]. They use Markov Chain Monte Carlo sampling to efficiently hone in on animations that meet all or most of the constraints. An example application of this simulation is the creation of multiple visually plausible animations that show a bowling ball knocking down only specified pins.

2.4.3 Timewarp Integration

Most techniques that find exact collision times advance and rewind the simulation of all bodies to obtain a chronological ordering for collisions. This method can become incredibly slow when many objects are colliding in different areas of the simulation. Brian Mirtich’s timewarp technique efficiently finds exact contact times for many body simulations [28]. Mirtich’s technique optimizes this process by only rewinding the state of objects that are near collision events. Since real-time applications need to synchronize the state of all objects many times a second for display purposes many of the benefits of this technique cannot be used, but some of the same ideas can still be applied.

2.5 Standard Real-Time Techniques

2.5.1 Penalty Force Response

The penalty force technique is a fairly simple technique that can produce reasonable results for both colliding and resting contact. This technique allows bodies to slightly interpenetrate, after which it applies a repellent “penalty” force proportional to the amount of penetration [37, 33, 30]. This technique is often not robust since different simulated situations may require varying time steps and penalty forces. The results are designed to look somewhat plausible, but the simulation basically treats object boundaries as very stiff springs instead of actual rigid surfaces. Another drawback to this approach is that a single collision can take multiple time steps to resolve. The penalty force technique is almost never used in interactive environments because a lack of robustness can lead to artifacts such as oscillating objects or rigid bodies popping through each other.

2.5.2 Impulse Based Response

Since applying impulses is usually fairly easy, Brian Mirtich proposed reducing all object interactions to instantaneous collisions [29]. Impulse based techniques require less parameter tuning than penalty force techniques, and objects do not interpenetrate. However, Mirtich reported that the simulation comes to a grinding halt when four boxes are placed on top of each other. This performance problem is due to the need for many impulses to propagate up and down the column of boxes before all collisions are resolved.

Guendelman et. al. recently expanded on Mirtich’s impulse based technique [20]. Their technique introduces an object space signed distance function for collision response, separate contact and collision impulses, and a final “shockwave” step where interpenetration

constraints are resolved.

2.6 Analytic Rigid Body Simulation

2.6.1 Original Formulation

Many of the contact models discussed so far have serious robustness problems when objects are in resting contact with one another. Analytic rigid body simulation aims to mathematically calculate and solve for the correct contact forces for multi-body contact configurations. In 1981 Per Lötstedt formulated the non-penetration constraint as a system of inequalities involving the forces at all contact points in the system [24]. From 1989 to 1995 David Baraff submitted a series of papers that created a framework for analytic rigid body simulations of this type [3, 4, 5, 6, 7, 8, 9]. Their technique finds a precise point of collision and analytically solves for the relative forces on all bodies. They pose force determination as a linear complimentary problem (LCP). Solving this LCP involves creating a matrix of values related to each contact point and repeatedly solving systems of equations within this matrix.

2.6.2 Gauss' Principle of Least Constraints

Redon et al. solve a system of inequalities that are based on Gauss' principle of least constraints, instead of the more straightforward non-penetration constraints [34]. This formulation solves explicitly for the degrees of freedom of the object, instead of for the magnitudes of the contact forces. Because of this formulation, the matrices which need to be solved have size proportional to the number of degrees of freedom of the object, (almost always six), instead of the number of contact points. Using this technique should be advantageous for simulations with an average number of contact points per object greater than six.

One of the difficulties in implementing this scheme is that it crafts the problem as a nearest-point problem (NPP). Although Redon et al. document the fact that they use Wilhelmson's method [38], in general there is far less literature related to solving nearest-point problems when compared to linear complimentary problems.

2.6.3 Position Based Time-Stepping

Analytic rigid body simulation using Baraff's formulation handles collision and contact interactions separately. This distinction necessitates starting and stopping the simulation

when instantaneous collisions occur. Stewart and Trinkle modify the analytic rigid body simulation technique to effectively integrate forces and resolve collisions at the same time [36]. The system of equations is reformulated to involve the integral of forces over a time step instead of the actual forces themselves. Using this method, contact and impulsive forces can be treated identically.

Schömer et. al. present a few variations for solving and formulating the above algorithm [35, 23, 21]. Anitescu and Potra extend this technique to always compute a valid solution regardless of the initial conditions [1, 2]. Implementation and other details are well explained in Michael Cline’s masters thesis [14].

2.6.4 Energy Based

Faure presented another technique whereby forces are determined in an iterative manner, allowing the user to make tradeoffs between accuracy and speed [18]. Faure’s technique iteratively adjusts contact forces and accelerations to more accurately reflect the conservation of energy. Because of the iterative nature of the algorithm, the user can make a tradeoff between accuracy and speed.

2.6.5 Articulated bodies

Articulated Bodies that have many fixed joints, such as a robot arm, may have many contacts but relatively few degrees of freedom. Articulated bodies also generally involve equality constraints instead of inequality constraints. By specifically designing for this situation, the system of equations can be greatly optimized. Baraff presented a method which solves a system of n loop free equality constraints in linear time [10]. Faure tackled the same problem, but his method allows for a tradeoff between accuracy and speed when solving cyclic constraints [19].

2.7 Position Optimizing Techniques

2.7.1 Potential Energy Minimization

Milenkovic greatly optimizes situations in which many objects are fall and come to rest [26]. Milenkovic’s technique simplifies the simulation by focusing on minimizing potential energy. Velocities are not stored per object, so this technique cannot handle free flying objects. It can, however, simulate objects that are falling due only to gravity. This technique excels

at simulating the falling or settling behavior of tightly bunched “clumps” of objects with many contact points.

2.7.2 Optimization Based Animation

Optimization based animation as presented by Milenkovic and Schmidt allows free flying objects and does not need to find exact collision times [27]. Instead of solving systems of equations using the standard Newtonian constraints they create artificial “goals” that they minimize. They then solve for positions that are close to the predicted Newtonian target positions but do not interpenetrate. After this they solve for a system of momentums that meet Newtonian constraints and minimize kinetic energy.

This technique can take arbitrarily large time steps but there are few drawbacks. After solving for the optimal new positions the configuration of objects will be touching in a large number of places. Furthermore quadratic programming (QP) techniques are needed which in general are slower than the linear complimentary problems (LCP) that many other analytic techniques use.

Chapter 3

Implementation

3.1 Simulation Domain

We restrict the domain of the simulation in a few different ways. We only deal with cuboid objects, commonly called oriented bounding boxes (OBBs) in computer graphics. By not considering general polyhedral models many of the collision detection and classification routines become easier to implement and faster to execute. Furthermore, some of our optimizations also take advantage of the fact that cuboids often come to rest on one of their six sides. Since the upcoming video game *Doom III* also seems to do rigid body simulation using only boxes, these choices seem reasonable for a class of real-time applications. Currently all boxes besides the ground plane have the same size and mass.

We allow certain objects to be marked as immovable, such as the ground plane. Many of our optimizations assume that we are dealing with a flat ground plane. Assuming a flat ground plane is generally true for many indoor environments, but may be less suitable for outdoor environments. The only external force we model is gravity. We do not model wind or air resistance.

3.2 Collision Detection

Collision detection is necessary for most physics simulations, but it is not our area of focus. For detecting intersections between oriented boxes, we use the fast separating axis technique by Eberly [16, 17]. This technique works well and also provides information about which faces are in contact.

To see whether two moving objects are colliding, we simply integrate the objects forward in time and check whether they are intersecting. Our approach assumes that objects cannot

intersect and separate in the span of one time step. This assumption does not hold for objects that briefly brush or for fast moving small objects. However, for many basic real-time applications, our approach seems to catch all visible collisions. Swept volume intersection tests can be used to more accurately test collisions for fast moving objects [28].

Analytic force determination requires the determination of the contact area for objects that are in resting contact and sharing a face-face contact. Since we are using OBBs, it is necessary to find the intersection of two oriented rectangles in 2D. Implementing this routine is not very difficult, but it is slightly tedious.

3.3 Collision and Contact Response

Collision and contact response is the main task of a physics simulator. When trying to come up with various techniques to optimize rigid body simulation, we experimented both with a physics engine primarily based on Baraff’s analytic force determination as well as the impulse based technique of Guendelman et. al.

3.3.1 Analytic Force Determination

In general, we found analytic force determination to be a very difficult foundation on which to build a general physics engine. This difficulty is mainly because Baraff’s method makes a distinction between collision and contact interactions, and because all collision interactions must be ordered chronologically.

Our analytic force determination algorithm is based on Baraff’s 1994 paper [8]. We needed to modify the code to account for some subtle cases due to loss of numerical precision. In the paper’s *computeForces* function there is a loop that continues to iterate until $a_d < 0$. We found it necessary to change this to $a_d < -\epsilon$ where ϵ is a small numerical tolerance (we are currently using 0.00001). Since this change allowed for small contact forces that actually push the objects together, we implemented a post-processing step that sets any negative forces to zero after the loop completes. Leaving out this post-processing step lead to results in which repeated time steps caused objects to gradually accelerate into each other until they completely passed through each other. We used Gaussian elimination to solve square matrices (a necessary step in Baraff’s algorithm).

Baraff’s technique calls for separate handling of collisions. We found the task of developing a physics engine that deals exclusively with instantaneous collisions to be a surprisingly subtle endeavor. Floating point precision problems as well as some slightly odd special cases can complicate a robust implementation. For instance, to find an exact collision time,

a common technique is to execute a binary search in the time dimension to find, within some small epsilon value, the exact instant when objects A and B go from separation to interpenetration. When completing this binary search, we may discover that we actually missed a collision between objects A and C. This discovery requires us to discard the current collision and restart the process focusing on the new collision point. Situations such as this case can be made less likely by taking smaller time steps, but discreet collision tests will never be able to catch all collisions.

Because of these difficulties, we were able to implement analytic force determination for contact interactions, and chronologically ordered Newtonian impulse determination for collision interactions, but we were not able to integrate the two schemes together. In our opinion, schemes that need to find exact collision times are at a disadvantage since the advance and retreat of time integration is costly in terms of both computation time and implementation complexity. In the future we would like to experiment with time stepping techniques that use many of the same systems that Baraff does, but treat collisions and contacts identically. Our investigation of chronologically ordered impulse simulations lead to the modified timewarp technique discussed in section 5.1 and appendix A.1.

3.3.2 Impulse Based

We found the impulse based technique presented by Guendelman et. al. to be much simpler to implement. Since impulse based techniques allow temporary interpenetration and treat collision and contact interactions identically, it was much easier to get the simulation to run robustly. However, immediately after our implementation was basically working, we noticed that our objects were not resting on the ground but oscillating wildly. These oscillations probably stemmed from a few different sources. The time step that we used was probably larger than the time steps used by Guendelman et. al., which could cause to minor instabilities. Also Guendelman et. al. used the gradient of an object space signed distance function to determine the “normal” for any point within an object. For speed reasons we decided not to use a signed distance function¹. We instead stored the closest face from the last time step and assumed that a vector perpendicular to this face would be a good approximation to the contact normal.

For instantaneous collisions, as opposed to resting contacts, we used the Newtonian collision model with a coefficient of restitution set globally to 0.9, (except when using the

¹Since we only used oriented boxes it should be possible to implement a relatively fast object space signed distance function.

sticky collision optimization, see section 4.3). Other more complicated collision models were available, such as the Poisson model, but we did not explore these methods. Collision interaction using the basic Newtonian model is covered very well by Witkin and Baraff's recent SIGGRAPH course notes [39]. The details of our impulse based method are discussed in section 5.2.

Chapter 4

Basic Techniques

4.1 Spatial Subdivision

Spatial subdivision is a useful technique for reducing the number of costly intersection tests. Using volumetric data structures to conservatively compute possible intersections is often called the broad phase of collision detection. Spatial subdivision can be very useful for physics simulations because objects tend to have both spatial as well as temporal coherence. Spatial and temporal coherence dictate that the boundaries of an object within space as well as the object's location within time tend to be localized. Spatial subdivision data structures such as kd-trees, octrees, axis-aligned hash tables [28], or axis-aligned hierarchical trees [31] are all useful and well documented techniques. We did not use spatial subdivision in our implementation because most of our test data sets consisted of objects interacting within in a very small space. For larger environments, spatial subdivision would become a necessity.

4.2 Visual Plausibility

Barzel et al. introduced the notion of visual plausibility, the idea that some slight amount of randomization within the simulation can make results seem more plausible [12]. If a cubic die lands on its side, we do not expect it to bounce straight up, but rather we expect there to be some slight variation in the outcome.

We incorporate this idea into our implementation by introducing randomness into high velocity collisions. For low velocity collisions and contact, we are more concerned that objects come to rest as quickly as possible. If we add randomness to low velocity collisions, we may introduce instability to the system that will keep the die continually jumping and spinning.

4.3 Sticky Collisions

As can be seen from the die example in the previous section, as one object comes to rest on another object it will undergo increasingly smaller and smaller collisions. In simulations that differentiate between contact and collision interactions, a number of collisions may need to be made before the simulation decides that the collisions are of a velocity small enough to describe the objects as being in resting contact. This chattering behavior can be very computationally expensive since the physics engine will only be able to integrate forward in very small steps. In this scenario smaller and smaller collisions occur in smaller and smaller time intervals. The chattering can also look unrealistic compared to the relatively few collisions real objects usually take to come to rest.

For this reason, if we are using analytic force determination and the relative velocity of two colliding objects is somewhat small (but still visibly colliding), we model the collision as a purely elastic collision so that the two objects “stick” together. The use of sticky collisions bypasses the numerous small collisions that normally occur before two objects are considered in resting contact. While one could model every collision as purely elastic and get the same savings, high impact collisions would then seem less realistic.

4.4 Fixing Interpenetration

It is often useful to have a process that manually fixes interpenetration without trying to maintain physical plausibility. Because round-off errors can cause interpenetration, manually enforcing non-penetration constraints is always a good idea. We use the shockwave technique of Guendelman et. al. by which objects are sorted and dealt with starting from the ground up along the y-axis [20]. Guendelman et. al. temporarily set lower objects to have infinite mass and then re-run their collision resolution algorithm. Since infinite mass objects cannot move, interpenetrating objects are pushed away. Since the technique is applied from the bottom up, generally objects get pushed upwards, (which makes sense because gravity causes objects to generally fall downwards). We modify the technique slightly by only changing the positions of objects instead of applying collision impulses. This change is motivated by our desire to avoid introducing new kinetic energy into the system. Since we generally move objects upwards, our technique does introduce potential energy into the system. Fixing interpenetration manually is obviously not physically accurate, but hopefully other more physically based methods will have already satisfied the vast majority of interpenetration constraints.

4.5 Computational Stability

If a group of objects has come to rest under gravity and nothing affects the group of objects, these objects will continue to stay at rest. Another way to phrase this idea is that if objects are at rest, they will not move until something makes them move. These groups of objects can be marked as stable and the simulator can henceforth assume that these objects will not move until a new interaction occurs. This extremely simple optimization can provide huge savings since users often interact with a small fraction of the entire environment at any one time. Using this technique, the physics engine will only need processing power when objects are in motion.

Chapter 5

Novel Techniques

5.1 Modified Timewarp

We do not use the full timewarp technique as presented by Mirtich [28], but we instead utilize the basic concepts of the method. Because real-time applications are forced to synchronize multiple times per second, many of the asynchronous benefits of the timewarp technique are lost. We use a modification of the timewarp technique in which all objects have only two states, a valid state and a test state. At the beginning of each frame, every object's state is integrated forward one time step and the information is stored in the test state. Collisions are then checked. If any collisions are found, exact collision times are determined for each colliding pair. The first colliding pair is determined, and the valid states of the colliding objects are updated. Finally, the list of collisions is updated given the new state of the two colliding objects.

We present two methods in pseudo code. The first method `SIMPLESYNCTIMEWARP` is the fairly straightforward algorithm. The second method `SAFERSYNCTIMEWARP` checks nearby neighbors to test if we have missed a nearby collision. However, `SAFERSYNCTIMEWARP` is still not guaranteed to order all events chronologically because the collision detection technique is discreet and therefore inexact. An example of this situation can be seen in figure 5.1. Since both methods may miss collisions and cause interpenetration, both methods call a final method that manually fixes interpenetration between objects.

Using swept volumes for collision detection will alleviate this problem. Swept volumes should be used if some objects travel at very high speeds relative to the integration time step. Since the swept volume technique is conservative, using swept volume will detect all possibly colliding pairs. However, discreet collision checks are usually used to finalize

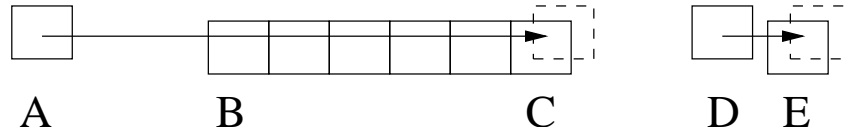


Figure 5.1: Object A has a high velocity moving right and is originally only detected to collide with object C. Object D has a small velocity moving right and originally is only detected to collide with object C. If D colliding with E is found to be the first collision, (updating their valid states), but the true sequence of events is $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$ our method will provide incorrect results. The use of swept volumes would detect the collision between A and B.

whether the objects collide or not. Because of the use of discrete checks, collision times can only be determined within a predefined margin of error ϵ . If multiple collisions occur within this margin of error, the collisions may be ordered incorrectly even when using swept volumes.

Since relying on a true chronological ordering of collisions due to numerical methods is not completely robust, any simulation that assumes interpenetration will not occur should manually enforce this constraint.

5.2 Damped Integration

While experimenting with the techniques of Guendelman et. al. [20], we found that some other techniques were necessary to eliminate oscillations of objects and achieve general stability. We introduce the *damped integration* scheme as a stable rigid body simulation technique for real time environments.

Our damped integration scheme builds upon the techniques of Guendelman et. al., discussed in section 2.5.2. Their method treats all interactions as instantaneous collisions that are resolved using impulses. First collisions are processed, and velocities are modified. Gravity is then integrated into the respective velocities for all objects. After this contacts are resolved much like collisions except a completely elastic collision model is used. Finally, the velocities of each object are used to update the object's final position.

We basically build on the foundation given by Guendelman et. al. We add a damping stage, a check for geometric stability, as well as a modified shockwave technique. Pseudo code for this technique is listed in appendix A.2.

5.3 Damping

We introduce a damping model that aims to produce friction like behavior and increase stability in the simulation. The model is physically inaccurate, but it appears that large liberties can be taken with frictional interactions while still maintaining visually plausible results. A greater concern when trying to maintain visual plausibility is making sure that energy is never added to the system. Not conserving energy can lead to very visible artifacts, such as chattering objects. Our damping model attempts to dissipate energy as quickly as possible while still maintaining plausible results.

The guiding principle behind this model is that friction slows things down. To combat the tendency of simulations to add energy to the system we only slow down or dampen objects. In Newtonian mechanics friction forces should be able to add kinetic energy (as in figure 5.2), but disregarding this effect does not drastically affect the plausibility of many animations. Since objects are acted upon independently, this method will not transfer friction forces along a chain of interacting objects.

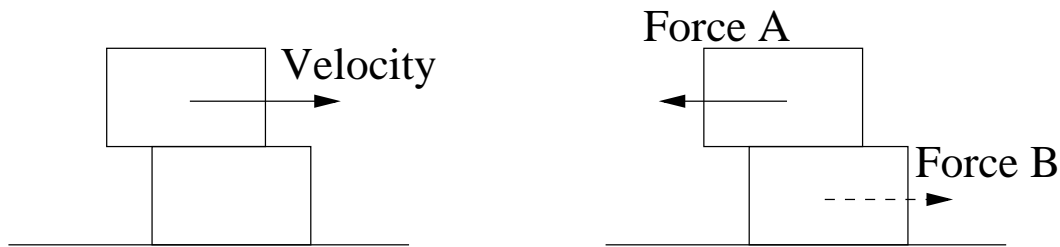


Figure 5.2: Using our damping method A will receive a damping effect close to the true Newtonian friction force. Force B should be applied according to Newtonian physics but will not be applied using our technique.

Our method is very simple: if two objects are found to be in resting contact, we scale down their velocities by a constant multiple. Decreasing velocity will unconditionally decrease kinetic energy, thereby stabilizing the simulation. This technique by itself is not enough to eliminate all chattering behavior. Our remaining techniques attempt to identify objects that are chattering and apply damping to these objects. The following attributes are common among chattering objects:

- In resting contact with another object
- Small linear and angular velocity
- Rotated so that a face is perpendicular to the direction of gravity

The last item is especially important when simulating very simple geometric objects in an environment with a flat ground plane. We can use the fact that boxes tend to come to rest on one of their sides as another indicator of whether an object is chattering. This indicator does not catch boxes that come to rest by leaning on other boxes.

We test for combinations of these three attributes and either scale down linear and angular velocities or set them to zero. For objects that are very close to having a face perpendicular to the direction of gravity, we may also rotate the object to bring it more in line with the direction of gravity. Our exact tests are given in appendix A.2.

We originally investigated the damping technique because of the savings that it can give to simulations that use analytic force determination. Newtonian friction can be modeled fairly accurately by modifying the system of equations that are solved during analytic force determination. However, adding friction to the system makes the corresponding matrix non-symmetric and less sparse. Because of the changes to the matrix including friction in the model necessitates the use of slower algorithms and more number crunching. Using our technique, one can instead solve the simpler matrix without friction and then use damping afterwards. Damping can be used because the damping technique is completely decoupled from collision and contact resolution. If our matrix is symmetric we can use a variety of faster algorithms such as the Cholesky based factorization method introduced by Davis and Hager [15]. David and Hager introduce a method for repeatedly solving sparse symmetric matrices that fits in perfectly with Baraff’s force determination paper [8].

5.4 Geometric Stability

A stable orientation is a configuration of objects that does not move due to gravitational force. Finding stable orientations has been discussed by Mattikalli et. al. [25], as well as Pang and Trinkle [32].

We introduce a new class of objects that are *geometrically stable*, assuming gravity along the negative y-axis and ignoring friction. This classification signifies that the object does not accelerate due to gravitational or contact forces no matter what their magnitude may be. If we know that the the object will not move no matter what contact forces are used, then we can exclude the model from these calculations.

We start classifying objects as geometrically stable from the bottom up (assuming that gravity is directed in the direction of the negative y-axis). An item is geometrically stable if it meets the following two requirements:

- The object has infinite mass (immovable) OR

- All contact normals are along the either the positive or negative y -axis AND
- Assuming that the object is a box¹, all four corners of the bottom plane of the box are in planar contact with other geometrically stable objects with a contact normal along the negative y -axis.

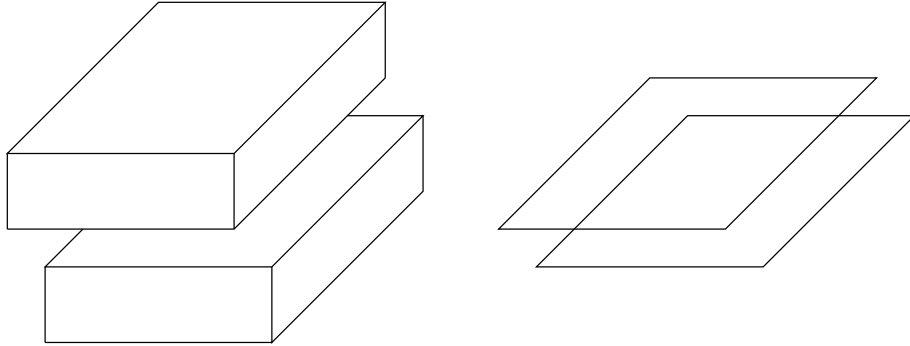


Figure 5.3: The top object is not geometrically stable. Only one of the corners in the top object is in contact with the bottom object. The left image shows the two objects in three dimensions and the right image shows the contact plane.

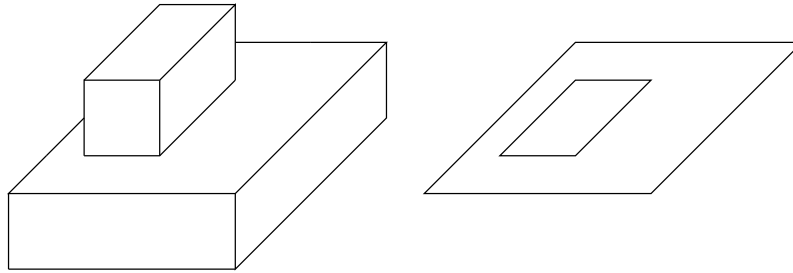


Figure 5.4: If the bottom object is geometrically stable then the top object is as well.

This technique saves a lot of computation time when calculating the contact forces for the system in figure 5.6. We do not need to find contact forces for any of the dark to dark box contact points because we know that the dark boxes do not accelerate due to contact forces. Where the dark boxes and light boxes touch we model the dark boxes as immovable and include the contact point in our system of unknowns. The exclusion of contact points

¹The more robust way of defining geometric stability for an object o is as follows. All contact points must either point along the positive or negative y -axis. Let all contact points with an outward surface normal that point along the negative y -axis be in set a . All contact points in set a must be interacting with another object n that is geometrically stable. Furthermore, the 2D convex hull surrounding the projection of these contact points into the xz plane must completely contain the projection of all of the object's surface points into the xz plane.

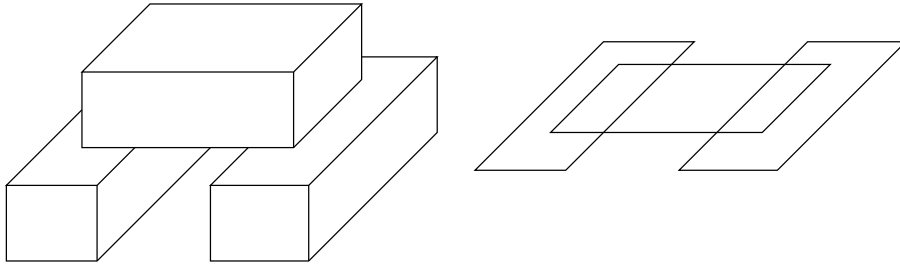


Figure 5.5: If both of the bottom objects are geometrically stable then the top object is as well.

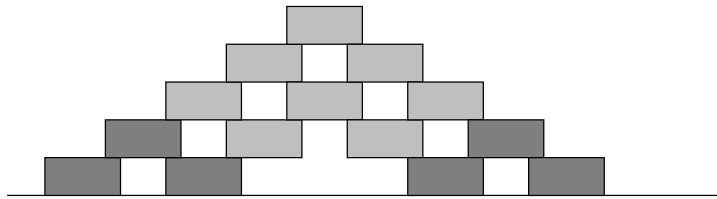


Figure 5.6: This figure shows a side view of a pyramid of objects where the bottom center object has been removed. The geometrically stable objects are colored dark grey where as the instable objects are colored light grey.

leads to a large performance gain because we must repeatedly solve matrices that have size proportional to the number of contact points squared.

Since geometrically stable objects can slide in the xz plane, we still apply damping to geometrically stable objects. Although contact forces are calculated assuming no friction, we may still have to use our friction model to modify the velocity of a geometrically stable object. A good example of this situation would occur in figure 5.4 if the top block had any velocity relative to the bottom block.

5.5 Filtering

The task of trying to eliminate chattering can be viewed as trying to eliminate very high frequency position and orientation changes. If we cannot eliminate chatter during the simulation, we can attempt to correct our data after the simulation is complete. We apply a low pass filter to our position and orientation data. Low pass filtering is a very common technique in the graphics and image processing community, but we have not seen it used within physical simulations.

We use a simple box filter to low pass filter each frame of a computed animation. The box filter covers the two neighboring frames of animation on each side and reduces chatter

significantly. The filtered results still show artifacts since objects that used to chatter will often appear to squirm instead. Widening the kernel width of our box filter cuts down on the squirming, but also eliminates other higher frequency data that we want, such as sharp collision response. Filtering data in this way also produces new non-physical results. The filtering process may create unwanted results, such as object interpenetration.

We also introduce an envelope filter that aims to more directly attack the chattering problem. We store an extra set of stable positions and orientations for each object, and only update these stable positions when the object moves outside of a spatial envelope surrounding its last stable position. In this way, fast moving objects are not filtered, and chattering objects that stay in the same general area are displayed as if they are not moving. The problem with this method is that objects may appear to pop when the stable positions and orientations are updated.

Filtering can be used in real time applications if the application stores previous states for objects. The simulation can then filter the previous states along with the current state and display the filtered result.

Chapter 6

Results

6.1 Damped Integration

Using our damped integration technique we were able to achieve visually plausible results in a real-time environment. In our interactive environment we were able to sling projectiles into clumps of objects as well as pyramid structures. Both of the simulations shown in figures ?? and 6.7 ran with frame rates above 20 frames per second on a 1500 MHz machine.

While we our techniques greatly improved the stability of the simulation, we were not able to totally eliminate object chattering. Our filtering techniques also helped eliminate artifacts, but either other minor artifacts like squirming (low pass filter), or popping (envelope filter).

6.2 Exterior Rotation Problem

After reading much of the literature for analytic force determination algorithms, one might be under the impression that integrating the calculated point forces prevent interpenetration. When actually attempting to simulate difficult situations, one finds that this is not true. The fact is that since we determine forces for an exact instant in time, integration of these forces may lead to incorrectly interpenetrating or separating objects.

This problem can be most readily seen when groups of objects undergo rotation. While the instantaneous contact forces are correct, integration of these forces using basic integration methods most likely leads to interpenetration¹. In figure 6.1, if there is a very high degree of static friction at all contact points, then all of the falling boxes stay connected

¹We have not tested on whether more accurate integration methods such as Runge-Kutta will accurately model a group of objects rotating around a single exterior point.

to each other and rotate around the contact point with the larger box. For the top box to follow the correct circular path, the force on the top box must be constantly changing. Since most simple integration schemes, such as the most basic forward Euler scheme used in our implementation, treat forces as constant over a single time step this will lead to incorrect results. Since force and torque are both measured with respect to the center of mass of the object, single values of force and torque cannot model an object undergoing circular motion around an exterior point for a non-zero amount of time.

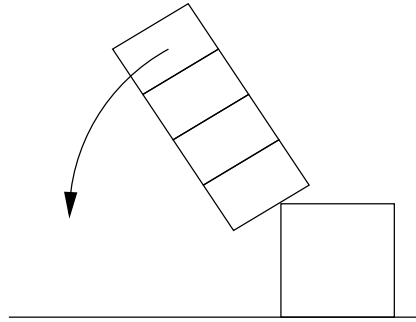


Figure 6.1: This figure shows that if the four smaller objects sticks together the top most object will undergo circular motion around an exterior point.

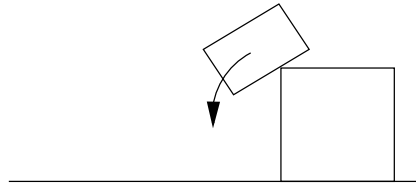


Figure 6.2: Although not as extreme, this situation will cause problems as well if static friction makes the smaller box rotate around the contact point instead of its own center of mass.

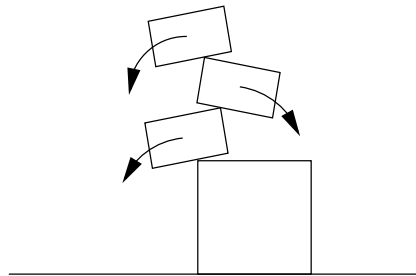


Figure 6.3: This figure shows an example of multiple rotational reference frames within one contact group.

To our knowledge, this problem has not been discussed in any detail. The problem is serious since interpenetration can lead to erroneous or missed collisions. To try to hide this problem, users may need to use much smaller time steps. Furthermore, while this rotation problem is highlighted by contact groups undergoing rotation, the problem still exists for single objects as in figure 6.2.

At first, it would seem that perhaps each contact group could store a pivot point and rotational velocity. One reference frame may not be good enough, however. We see from figure 6.3 that when dealing with edge contacts, we can have multiple rotating reference frames within group of objects.

Rotational reference frames within rotational reference frames do not simplify like inertial reference frames do. Besides more data structure headaches, many of the calculations get ugly fairly quickly. Centripetal forces must be used and moment of inertia calculations may have to be modified.

6.3 Future Work

6.3.1 Novel Techniques

With further study we believe that our damping method could be refined to more closely model Newtonian friction. We would also like to make the model more general, since our coefficients currently assume a specific size and mass for all objects.

We would also like to experiment with other ways of detecting stability. Dynamically reducing time steps could help reduce chatter and more quickly identify and determine stable states for objects. Classifying more objects as stable in turn improves performance, since stable objects are not fully simulated due to our computational stability technique. We would also like to look into using data from previous time steps to determine whether an object is chattering.

The geometric stability classification we presented is fairly narrowly defined. It would seem that we should be able to extend this technique to simplify situations such as figure 6.4. However, more complicated situations such as figure 6.5 have so far discouraged us from a more general approach.

6.3.2 Rigid-Body Research

The amount of code publically available for research purposes in rigid body simulation is minimal. We plan to release the source code of our implementation after it has been

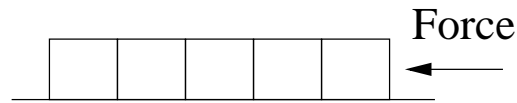


Figure 6.4: It should seem that using geometric simplification this series of blocks could be treated as one heavier object.

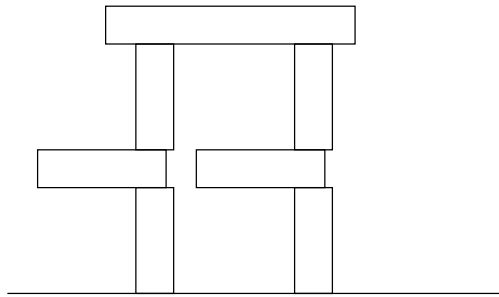


Figure 6.5: If the top block is very heavy relative to the blocks below it this configuration will be stable. The fact that the weight of the top block is distributed between two separate contacts makes this problem difficult.

tuned and slightly reorganized. We hope that by releasing the source code we can help others identify and avoid common pitfalls, such as numerical tolerances, that are not often discussed in the relevant literature.

A performance evaluation of optimized implementations of the many techniques is also lacking. Better research with regards to relative performance is probably necessary before software companies are willing to integrate rigid body simulation into their products. Ideally, the code used for this performance evaluation would be released under a non-restrictive license so that others can view and modify the code and move this area of research forward at a faster rate.

6.4 Conclusion

We have summarized various techniques for rigid body simulation. We have also discussed both basic and novel techniques for this area of research. Finally, we have produced a simulator that incorporates our novel techniques to achieve rigid body simulation in real-time.

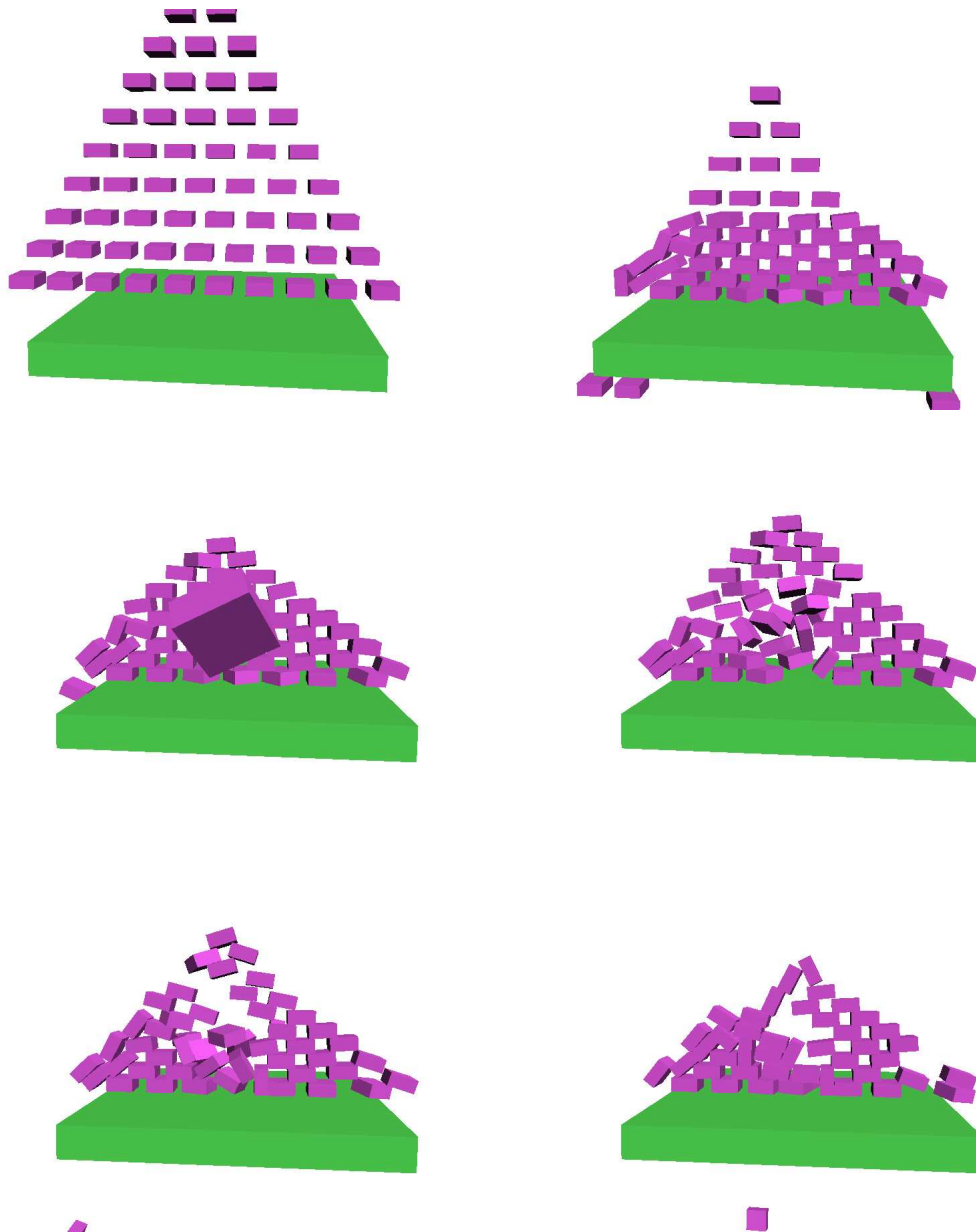


Figure 6.6: In this interactive simulation a box is shot out from the eye point towards the pyramid. This simulation ran at 25 fps on a 1500 MHz AMD processor

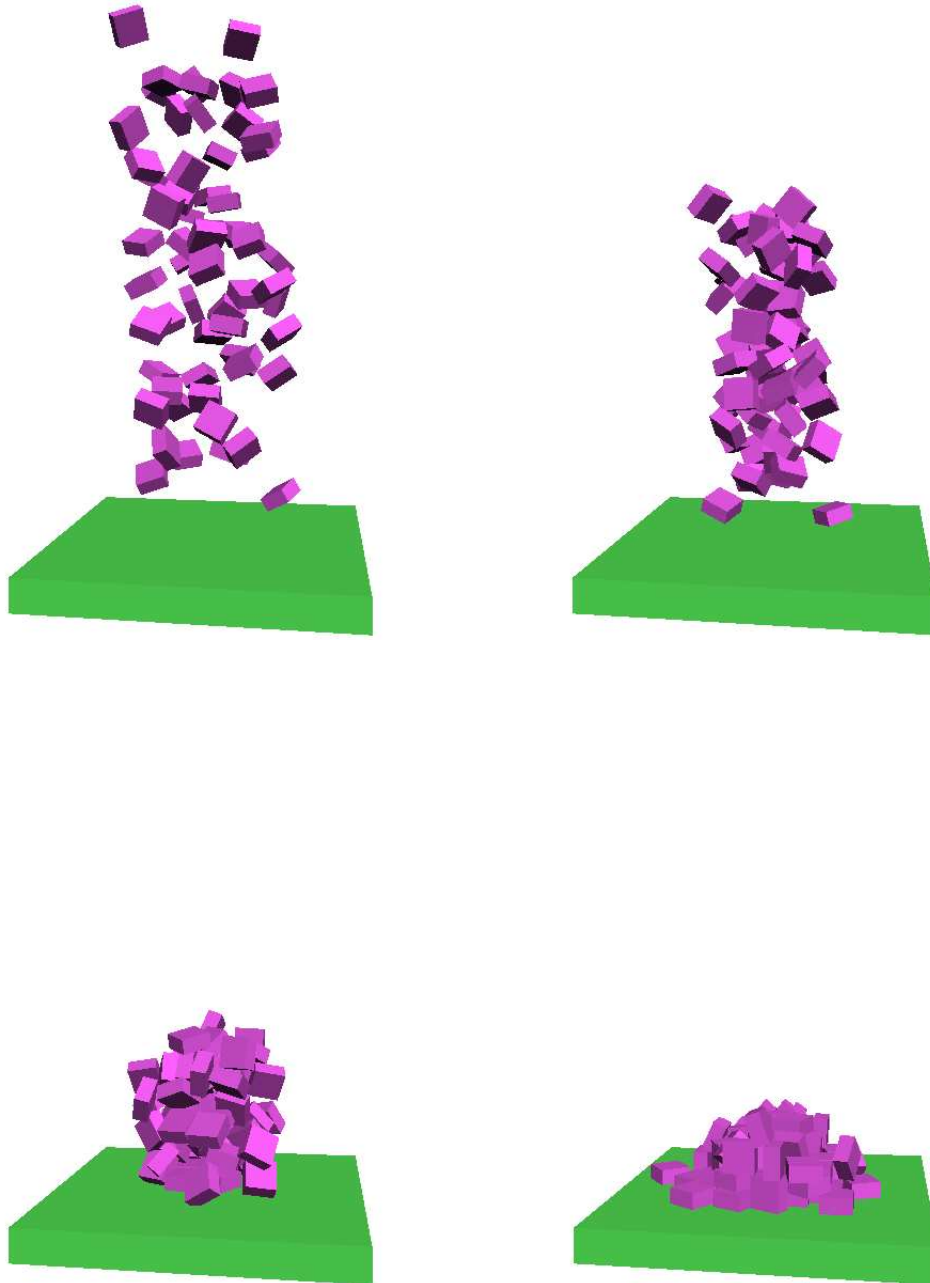


Figure 6.7: In this case all objects had an initial velocity towards the origin. After the objects settle some small chatter is still found. This simulation ran at 20 fps on a 1500 MHz AMD processor.

Appendix A

Pseudo Code

A.1 Modified Timewarp

```
SIMPLESYNC TIMEWARP(startTime,  $\Delta t$ , resolution)
1  for i  $\leftarrow$  1 to numObjects
2    testObjects[i]  $\leftarrow$  validObjects[i]
3    integrateForward(testObjects[i],  $\Delta t$ )
4    validTimes[i]  $\leftarrow$  startTime
5  collisionHeap  $\leftarrow$  findAllCollisions(testObjects, resolution)
6  while collisionHeap.isEmpty() = FALSE
7    curCollision  $\leftarrow$  collisionHeap.removeFirstCollision()
8    for i  $\leftarrow$  1 to 2
9      index  $\leftarrow$  curCollision.objectIndices[i]
10     removeCollisionsForObject(collisionHeap, index)
11     integrateForward(validObjects[index], curCollision.time - validTimes[index])
12     validTimes[index]  $\leftarrow$  curCollision.time
13     collideTwoObjects(validObjects, curCollision)
14     for i  $\leftarrow$  1 to 2
15       index  $\leftarrow$  curCollision.objectIndices[i]
16       testObjects[index]  $\leftarrow$  validObjects[index]
17       integrateForward(testObjects[index], (startTime +  $\Delta t$ ) - curCollision.time)
18       addCollisionsForObject(testObjects[index], resolution, collisionHeap)
19  for i  $\leftarrow$  1 to numObjects
20    validObjects[i]  $\leftarrow$  testObjects[i]
21    validTimes[i]  $\leftarrow$  (startTime +  $\Delta t$ )
22  fixInterpenetration()
23  return
```

```

SAFERSYNCTIMEWARP(startTime, Δt, resolution)
1  for i ← 1 to numObjects
2    testObjects[i] ← validObjects[i]
3    integrateForward(testObjects[i], Δt)
4    validTimes[i] ← startTime
5  collisionHeap ← findAllCollisions(testObjects)
6  while collisionHeap.getSize() > 0
7    curCollision ← collisionHeap.removeFirstCollision()
8    nearIndices ← findNearObjects(testObjects, curCollision)
9    for i ← 1 to nearIndices.getSize()
10     index ← nearIndices[i]
11     testObjects[index] ← validObjects[index]
12     integrateForward(testObjects[index], curCollision.time - validTimes[index])
13  tempHeap ← findCollisionsForGroup(testObjects, nearIndices)
14  if tempHeap.getSize() > 0
15    collisionHeap.insert(tempHeap.removeFirstCollision())
16  else
17    for i ← 1 to 2
18      index ← curCollision.objectIndices[i]
19      nearIndices.remove(index)
20      removeCollisionsForObject(collisionHeap, index)
21    for i ← 1 to nearIndices.getSize()
22      testObjects[index] ← validObjects[index]
23      integrateForward(testObjects[index], curCollision.time - validTimes[index])
24  collideTwoObjects(testObjects, curCollision)
25  for i ← 1 to 2
26    index ← curCollision.objectIndices[i]
27    testObjects[index] ← validObjects[index]
28    integrateForward(testObjects[index], (startTime + Δt) - curCollision.time)
29    addCollisionsForObject(testObjects[index], resolution, collisionHeap)
30  for i ← 1 to numObjects
31    validObjects[i] ← testObjects[i]
32    validTimes[i] ← (startTime + Δt)
33  fixInterpenetration()
34  return

```

A.2 Damped Integration

DAMPEDINTEGRATION(Δt)

```

1  ResolveCollisions( $\Delta t$ )
2  ResolveContacts( $\Delta t$ )
3  IntegrateObjectVelocities( $\Delta t$ , STANDARD_GRAVITY)
4  DampenObjects()
5  FixContactConstraints()
6  SetGeometricallyStable()
7  return
```

COLLIDE(collision, restitution, gravity)

```

1  if gravity = VELOCITY_PLUS_GRAVITY
2      deltaVelocity = GRAVITY_VECTOR
3  else
4      deltaVelocity = ZERO_VECTOR
5  for i  $\leftarrow$  1 to 2
6      collision.getObject(i).velocity  $\leftarrow$  collision.getObject(i).velocity + deltaVelocity
7  findContactPoint(collision)
8  applyNewtonianImpulse(collision, restitution)
9  for i  $\leftarrow$  1 to 2
10     collision.getObject(i).velocity  $\leftarrow$  collision.getObject(i).velocity - deltaVelocity
11     if collision.getObject(i).geometricallyStable
12         collision.getObject(i).linearMomentum[1]  $\leftarrow$  0
13         collision.getObject(i).angularMomentum[0]  $\leftarrow$  0
14         collision.getObject(i).angularMomentum[2]  $\leftarrow$  0
15 return
```

RESOLVECOLLISIONS(Δt)

```

1  for i  $\leftarrow$  1 to NUM_COLLISION_ITERATIONS
2      integrateObjectPositions( $\Delta t$ , VELOCITY_PLUS_GRAVITY)
3      gatherCollisions()
4      for c  $\in$  collisions
5          collide(c, COLLISION_RESTITUTION, VELOCITY_NO_GRAVITY)
6      revertToOriginalPositions()
7  return
```

RESOLVECONTACTS(Δt)

```

1  for i  $\leftarrow$  1 to NUM_CONTACT_ITERATIONS
2    integrateObjectPositions( $\Delta t$ , VELOCITY_PLUS_GRAVITY)
3    gatherCollisions()
4    for c  $\in$  collisions
5      collide(c, -1 + (i / NUM_CONTACT_ITERATIONS), VELOCITY_PLUS_GRAVITY)
6    revertToOriginalPositions()
7  return

```

DAMPENOBJECTS(Δt)

```

1  for o  $\in$  objects
2    if o has had a contact interaction
3      yAxisAligned  $\leftarrow$  checkYAxisAlignment()
4      o.linearMomentum  $\leftarrow$  o.linearMomentum *  $k_1$ 
5      o.angularMomentum  $\leftarrow$  o.angularMomentum *  $k_2$ 
6      if yAxisAligned
7        if isSmall(o.linearMomentum[1],  $\epsilon_3$ )
8          o.linearMomentum[1] = 0;
9          o.angularMomentum = o.angularMomentum *  $k_3$ ;
10       tiltTowardsYAxis(o,  $\epsilon_4$ )
11      for i  $\leftarrow$  1 to 3
12        if isSmall(o.linearMomentum[i],  $\epsilon_5$ )
13          o.linearMomentum[i] = o.linearMomentum[i] *  $k_5$ ;
14        if isSmall(o.linearMomentum[i],  $\epsilon_5$ )
15          o.linearMomentum[i] = 0;
16        if isSmall(o.angularMomentum[i],  $\epsilon_7$ )
17          o.angularMomentum[i] = o.angularMomentum[i] *  $k_7$ ;
18        if isSmall(o.angularMomentum[i],  $\epsilon_8$ )
19          o.angularMomentum[i] = 0;
20        if isSmall(o.angularMomentum[1],  $\epsilon_9$ ) AND (o.angularMomentum[1]  $\geq$  0)
21          o.angularMomentum[1] = 0;
22      integrateObjectPositions( $\Delta t$ , STANDARD_GRAVITY)
23  for o  $\in$  objects
24    deltaPosition  $\leftarrow$  length(o.newPosition - o.oldPosition)
25    deltaRotation  $\leftarrow$  length(o.newRotation - o.oldRotation)
26    if isSmall(deltaPosition,  $\epsilon_{10}$ ) AND isSmall(deltaRotation,  $\epsilon_{11}$ )
27      o.newPosition  $\leftarrow$  o.oldPosition
28      o.newRotation  $\leftarrow$  o.oldRotation
29  return

```

FIXCONTACTCONSTRAINTS()

```

1 sortedContacts ← sortContactPointsAscending()
2 for i ← 1 to sortedContacts.getSize()
3   movingObject ← sortedContacts[i].getUpperObject()
4   stableObject ← sortedContacts[i].getLowerObject()
5   offsetVector ← sortedContacts[i].getLowerToUpperNormal()
6   while intersecting(movingObject, stableObject)
7     movingObject.center ← movingObject.center + offsetVector *  $\epsilon_1$ 
8 return

```

SETGEOMETRICALLYSTABLE()

```

1 for o ∈ objects
2   if o.mass =  $\infty$ 
3     o.geometricallyStable ← TRUE
4   else
5     o.geometricallyStable ← FALSE
6 sortedContacts ← sortContactPointsAscending()
7 for i ← 1 to sortedContacts.getSize()
8   c ← sortedContacts[i]
9   topIndex ← c.getTopObjectIndex()
10  botIndex ← c.getBottomObjectIndex()
11  o ← c.getObject(topIndex)
12  outwardNormal ← c.getOutwardNormal(topIndex)
13  if equivalent(outwardNormal, NEGATIVE_Y_AXIS) AND (o.getObject(botIndex).geometricallyStable)
14    projectedContactPoint ← projectXZPlane(c.point)
15    o.markProjectedCorners(projectedContactPoint)
16    if o.allCornersMarked()
17      o.geometricallyStable ← TRUE
18  else
19    if equivalent(outwardNormal, POSITIVE_Y_AXIS) = FALSE
20      o.geometricallyStable ← FALSE
21 return

```

Bibliography

- [1] M. Anitescu and F. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14:231–247, 1997.
- [2] M. Anitescu and F. Potra. A time-stepping method for stiff multibody dynamics with contact and friction. *International Journal of Numerical Methods in Engineering*, 55(7):753–784, 2002.
- [3] David Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics*, 23(3):223–232, 1989.
- [4] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics*, 24(4):19–28, 1990.
- [5] David Baraff. Coping with friction for non-penetrating rigid body simulation. *Computer Graphics*, 25(4):31–40, 1991.
- [6] David Baraff. *Dynamic Simulation of Nonpenetrating Rigid Bodies*. PhD thesis, Cornell University, 1992.
- [7] David Baraff. Issues in computing contact forces for nonpenetrating rigid bodies. *Algorithmica*, 10:292–352, 1993.
- [8] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. *Computer Graphics*, 28(Annual Conference Series):23–34, 1994.
- [9] David Baraff. Interactive simulation of solid rigid bodies. *IEEE Computer Graphics and Applications*, 15(7):63–75, 1995.
- [10] David Baraff. Linear-time dynamics using lagrange multipliers. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 137–146. ACM Press, 1996.

- [11] David Baraff and Andrew Witkin. Large steps in cloth simulation. *Computer Graphics*, 32(Annual Conference Series):43–54, 1998.
- [12] Ronen Barzel, John F. Hughes, and Daniel N. Wood. Plausible motion simulation for computer graphics animation. *Computer Animation and Simulation*, Proceedings of the Eurographics Workshop:183–197, 1996.
- [13] Stephen Chenney and D. A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 219–228, 2000.
- [14] Michael Cline. Rigid body simulation with contact and constraints. Master’s thesis, The University of British Columbia, 2002.
- [15] Timothy A. Davis and William W. Hager. Modifying a sparse Cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 20(3):606–627, 1999.
- [16] David Eberly. Dynamic collision detection using oriented bounding boxes. <http://www.magic-software.com/>.
- [17] David Eberly. Intersection of objects with linear and angular velocities using oriented bounding boxes. <http://www.magic-software.com/>.
- [18] François Faure. An energy-based method for contact force computation. In *Computer Graphics Forum (Proceedings of EUROGRAPHICS’96)*, volume 15, pages 357–366, Aug 1996.
- [19] François Faure. Fast iterative refinement of articulate solid dynamics. In Hans Hagen, editor, *IEEE Transactions on Visualization and Computer Graphics*, volume 5, pages 268–276, 1999.
- [20] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. *Computer Graphics*, 2003, to appear.
- [21] G. Hotz, A. Kerzmann, C. Lennerz, R. Schmid, E. Schmer, and T. Warken. Calculation of contact forces. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 180–181. ACM Press, 1999.
- [22] Thomas Jakobsen. Advanced character physics. Proceedings, Game Developer’s Conference, San Jose, 2001.

- [23] C. Lennerz, E. Schöemer, and T. Warken. A framework for collision detection and response. In *11th European Simulation Symposium*, volume 2, pages 309–314, 1999.
- [24] Per Lötstedt. Coulomb friction in two-dimensional rigid body systems. *Zeitschrift für angewandte Mathematik und Mechanik*, 61:605–615, 1981.
- [25] Raju Mattikalli, David Baraff, and Pradeep Khosla. Finding all gravitationally stable orientations of assemblies. In *Proceedings of the IEEE International Conference of Robotics and Automation*, volume 1, pages 251–257, 1994.
- [26] Victor J. Milenkovic. Position-based physics: simulating the motion of many highly interacting spheres and polyhedra. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 129–136. ACM Press, 1996.
- [27] Victor J. Milenkovic and Harald Schmidl. Optimization-based animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 37–46. ACM Press, 2001.
- [28] Brian Mirtich. Timewarp rigid body simulation. *Computer Graphics*, pages 193–200, 2000.
- [29] Brian Mirtich and John F. Canny. Impulse-based simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*, pages 181–188, 217, 1995.
- [30] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(Annual Conference Series):289–298, 1988.
- [31] James F. O’Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. *Computer Graphics*, pages 111–120, 1999.
- [32] John-Shi Pang and J. C. Trinkle. Stability characterizations of rigid body contact problems with coulomb friction. *Zeitschrift für angewandte Mathematik und Mechanik*, 80(10):643–663, 2000.
- [33] J.C. Platt and A.H. Barr. Constraint methods for flexible models. *Computer Graphics*, 22(Annual Conference Series):279–288, 1988.
- [34] S. Redon, A. Kheddar, and S. Coquillart. Gauss’ least constraints principle and rigid body simulation. In *IEEE International Conference on Robotics and Animation*, May 2002.

- [35] Jörg Sauer and Elmar Schömer. A constraint-based approach to rigid body dynamics for virtual reality applications. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology 1998*, pages 153–162. ACM Press, 1998.
- [36] David Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with coulomb friction. *Zeitschrift für angewandte Mathematik und Mechanik*, 77(4):267–279, 1997.
- [37] D. Terzopoulos, J. C. Platt, A.H. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics*, 21(4):205–214, 1987.
- [38] D. R. Wilhelmsen. A nearest point algorithm for convex polyhedral cones and applications to positive linear approximations. *Mathematics of Computation*, 30:45–57, 1976.
- [39] Andrew Witkin and David Baraff. Physically based modeling. *Computer Graphics*, Course Notes, 2001.
- [40] Richard Wyckoff. Postmortem: Dreamworks interactive’s trespasser. *Game Developer*, June 1999.